A Deep Learning Solution for Blood Diagnostics of Cancers through Error Suppression

Zaid Nabulsi (znabulsi) Vineet Kosaraju (vineetk) Shuvam Chakraborty (shuvamc)

1. Introduction

Blood diagnostics of cancer present a golden opportunity to advance the state of cancer treatment and are made possible due to the presence of circulating tumor DNA. However, diagnosing cancer through the blood is an extremely difficult task that is analogous to finding a needle in a haystack, since the *vast* majority of a patient's DNA found in the bloodstream is from healthy, non-cancerous cells with an estimated one tumorous DNA fragment for every 500,000 DNA fragments. This challenge is compounded by the fact that when DNA is sequenced, there is a large number of human-introduced errors brought about by mistakes in the sequencer, among many other reasons. In fact, a large majority of of non-reference bases found in a cancer patient's sequenced genome come from human-introduced errors, and are not biological. Thus, to make the task of blood diagnostics of cancer feasible, there must be a clear method to classify a given non-reference base as either a human-introduced error, or as a biological mutation that may indicate cancer. Here, we present such a binary classifier using a deep learning model that will accurately predict whether a single non-reference base is human introduced or whether it may be an indicator of disease. The input is a non-reference base from a BAM file (from which we extract relevant features and run through our model), and the output is a binary classification (0 for human-introduced error, 1 for biological mutation). This will pave the way to future blood diagnostics of cancer, which can have revolutionary consequences on the state of cancer treatment.

2. Related Work

While introduced errors in the genome is a widely-known problem, this particular topic is highly specific and lacks much, if any, machine learning based prior research. There has been prior work done by Illumina with general hardware genome sequencing techniques to limit errors[1]. For correcting sequencing errors, current models use a probabilistic technique depending on the sequence of base pairs around the suspected errors[2]. Nevertheless, we perused general machine learning work in genomics to draw valuable intuition for our models, e.g. how to take advantage of the structure of genetic data, such as Andrew Ng's work with ECG's[3], general applications to genetics[4], and the importance of machine learning in functional genomics[5].

3. Dataset and Features

The Alizadeh Lab in the Stanford School of Medicine has collected genome sequences from both healthy patients as well as patients with cancer. The dataset consists of this patient data, where each patient has three billion base pairs. The dataset comes in raw FASTQ files outputted from an Illumina sequencer, but we convert these to BAM files to extract features. A full listing of features is shown in Table 1. Extracting features from BAM files gives us a processed dataset where each training example refers to a single, *non-reference* base pair, and the label for the example refers to whether that base pair is a non-biological error (that should be error-corrected, or "polished", a 0), or a biological error (a 1). We accumulated over 30 million training examples from roughly 300 different patients. Due to the nature of the problem, our data is rather skewed: about 86.3 percent of our training examples are labelled class 0. To help with this data skew, we use F1 score as our main evaluation metric, and oversample from the class 1 examples when training. We use the same 90/5/5 train/dev/test split across all our models for consistency.

Feature Name	Feature Description	Feature Name	Feature Description
Allele Frequency	Relative freq. of allele at locus	# Non-Ref Bases	# of non reference bases on the read.
Barcode Family	# of PCR duplicates generated	Base Quality	The PHRED quality of the base.
Base Change	A constant (0-11)	Mapping Quality	The mapping quality of the read
Duplex	Does it come from duplex mol.	Fragment Length	The length of the fragment.
Read 1/2	Binary; read is Watson or Crick	Polishing P-Value	The p-value based off a database of healthy people.

Plus / Minus	Binary; plus or minus strand	Polish Normally	Binary; would base get polished out or not
Pos on Read	Decimal for the base pos (0-1).	Type of Cancer	Which cancer type the base is from.

Table 1: *The 14 features extracted from the raw genome sequencing data files being used for the binary classification task.*

Methods

In order to evaluate the neural network models we developed and compare their results to the state-of-the-art techniques being used in the field, we initially implemented a variety of baselines. We then implemented three core main models: "DeepNet", "TwoNet", and "ThreeNet", discussed below. For these three models we performed an extensive hyperparameter search varying various parameters such as batch size and learning rate, as well as the number of layers and nodes per layer. For each model we optimize cross-entropy loss, defined for our two-class classification as follows where h_θ represents the model:

$$l(\theta) = \sum_{i=1}^{n} y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)})).$$

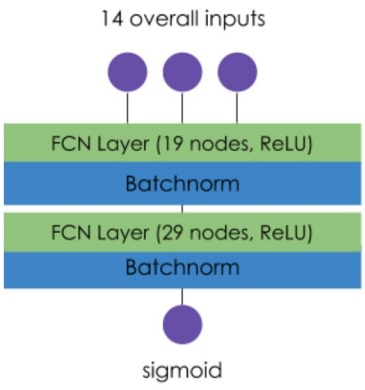
4.1 Baseline Model

To establish baseline performance we used three models. The first was a statistical framework model (an existing technique developed by the Alizadeh lab). This is performed in the following manner: start with a background database of relevant positions in the genome with a cohort of 12 healthy patients with statistics about the relevant base positions (how often base-specific base changes at each position occur). Next, when looking at the patient, p-values are generated for every non-reference base and compared to a Bonferroni corrected threshold. If the p-value is statistically significant, the error is attributed to a biological cause. Otherwise, the error is attributed to noise/human error. The second baseline was a logistic regression, and the third was a shallow neural net with one hidden layer of size 16 with ReLU activation and a sigmoid activated output layer with hyperparameter tuning of the learning rate, hidden layer size, and activation. Baselines were trained on the oversampled train set and evaluated with the F1 metric on the train and dev sets. The results are as follows:

Metric	Train Accuracy	Train Precision	Train Recall	Train F1 Score	Dev Accuracy	Dev Precision	Dev Recall	Dev F1 Score
Statistical Framework	0.883245	0.456595	0.504761	0.479472	0.883284	0.456390	0.505119	0.479520
Logistic Regression	0.90412	0.52361	0.50765	0.51550	0.90421	0.52351	0.50752	0.51391
2 Layer NN	0.91643	0.53461	0.51089	0.52248	0.91632	0.53426	0.51078	0.53461

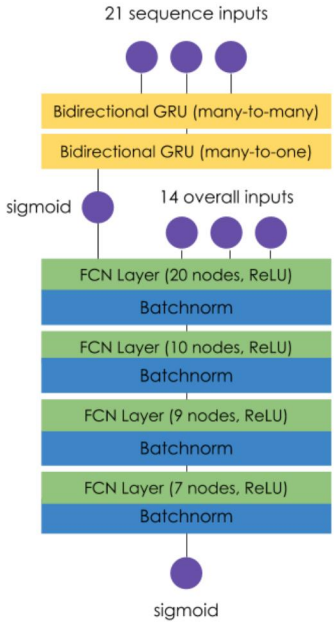
As expected, the baseline results are suboptimal, as these models do not have enough complexity to generate a good decision boundary, though these baseline results were still encouraging as increasing complexity gave better results.

4.2 Model 1: "DeepNet"



The first model we implemented consisted of a deep neural network that took as input the 14 core features (*Dataset and Features)*, and outputted a binary prediction (0/1) for the class. The detailed architecture is shown to the left in Figure 1. Each node in each fully connected layer is activated by the Rectified Linear Unit (ReLU) function, and these activations are passed through a batch normalization layer to speed up training and make the model more robust to poor weight initializations. The final layer is passed through a sigmoid activation to ensure that predictions remain between 0 and 1. The number of hidden layers (two) and number of nodes in each layer (19 followed by 29), were chosen using a random search for hyperparameter optimization on the dev set[6].

4.3 Model 2: "TwoNet"



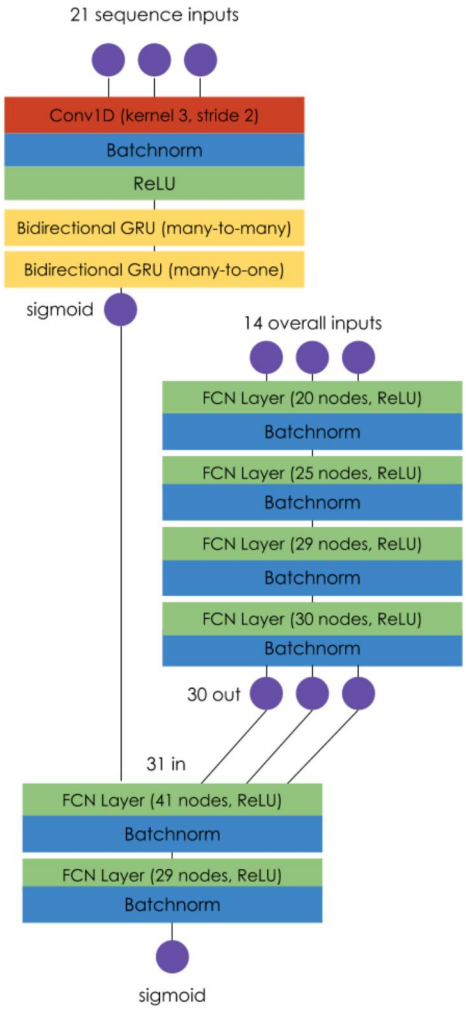
The second model we implemented consisted of two main networks: a RNN that utilized time-series sequence data, as well as a deep neural network that outputted a binary prediction. Since genomic nucleotides are inherently in-sequence, with the placement of bases and neighbors serving as important features for understanding genetic data, we additionally used this sequence information in our models, providing the motivation for developing this *TwoNet*. For instance, the sequencer may get thrown off by certain sequence patterns. Specifically, we extracted 21 inputs for each training example composed of the 10 bases before and after the current base. These bases are fed into a RNN composed of two bidirectional GRU layers. The layers are made bidirectional as the predictions depend on neighbors both before and after the current base. Further we use two layers as we hope that the first learns an encoding for the sequence input, while the second decodes that sequence into a feature for the second part of the network, a tactic applied in machine translation tasks that we empirically verified. Note that the output encoding passes through a sigmoid; while is this rather unconventional and not necessary, we noticed empirically that it resulted in the best models. The choice of GRUs over vanilla RNNs and LSTMs was also chosen during the search process. The second part of the model is similar to the *DeepNet*, where the main differences are the additional sequence input, the number of hidden layers (four), and the nodes per layer (20-10-9-7), which were chosen by random hyperparameter search (Figure 2, left).

4.4 Model 3: "ThreeNet"

The third and most complex model we implemented consists of two encoder networks that fed their learned outputs into a deep network to perform the binary classification task (Figure 3, right).

The first encoder network encodes the 21 sequence inputs into one sequence information output, similar to *TwoNet*, with the main difference the one dimensional convolutional layer before the RNN layers. This convolutional layer converts the 21 sequence inputs into 10 inputs for the dual purposes of reducing the dimensionality of inputs for the RNN, as well as to learn similar features for neighboring bases. This layer is used frequently in machine translation architectures; a problem similar to the task of learning an encoding for the genetic sequence information. The outputs from this layer are fed into a batch normalization layer and then through a ReLU activation to speed up the learning process.

The second encoder network encodes the 14 overall feature inputs into a vector of length 30 through several fully connected layers. This encoding network is similar to that of *DeepNet*, but with the final output layer removed, and the number of layers and nodes per layer set to different hyperparameter settings. The creation of this encoder network provides the main motivation for the *ThreeNet*. The third component of the network consists of a deep neural network that takes as input the 31 encoded features and outputs a binary prediction (0/1) for the class. This network is also similar to that of the *DeepNet*, with the number of hidden layers (two) and the number of nodes (41-29) chosen using random search.



5. Experiments, Results & Discussion

We ran experiments on our three core models of the *DeepNet*, *TwoNet*, and *ThreetNet*, and report results after extensive tuning in Table 3. As in the baselines, we record precision, recall, and F1 score, and use F1 score as our main metric in optimizing our models, due to our data skew. Due to this skew, accuracy does not provide a meaningful measure, so we do not report it.

In the *DeepNet*, we conducted an extensive hyperparameter search, experimenting with different learning rates, batch sizes, activation functions, and optimization techniques. After testing a few different learning rates, it turned out that the learning rate that converged the best was 1e-4. Larger numbers appeared to result in oscillations, while smaller numbers did not converge. For batch size, we wanted to choose the largest batch size that would fit in memory in order to allow us to take bigger step-sizes. For that reason, we found a batch size of 512 to be fitting. For our activation functions, we experimented with using sigmoid, TanH, ReLU, and Parametric ReLU for our hidden layers. Through our experiments, we discovered that the sigmoid function converged the slowest, likely due to the vanishing gradient problem, and that TanH converged only marginally faster. We found our best success with a ReLu activation and empirically confirmed that we did not have a "dead neuron" problem, removing the need for Leaky or Parametric ReLu, which performed similarly. For the sake of simplicity, we decided to move forward with the regular ReLu function for our hidden layers. For our output layer, we consistently used a sigmoid, since we are performing a binary classification. Also as part of our extensive hyperparameter search, we tried out different optimization algorithms including mini-batch gradient descent, momentum, Nesterov Accelerated Gradient Descent[7], and Adam. As expected, we found that mini-batch gradient descent and momentum took a substantial amount of time to converge, due to their inability to efficiently navigate saddle points and slow down when approaching minima. Although we expected Nesterov's to better navigate areas close to minima[7], we only found slight improvements in convergence and so we used Adam, which converged the fastest and had the most optimal performance due to its ability to address vanishing gradients and high variances in parameter updates.

We report a summary of our accuracies for our *DeepNet* in Table 3 below. We were able to achieve a very high precision value at roughly 0.975, but our recall was significantly lower, at 0.73. We also noted that our train results were very similar to our dev results, indicating that overfitting was not at all an issue and that we should be able to use more complicated models to fit the training set, something we saw repeatedly with our *TwoNet* and *ThreeNet*. In our *TwoNet* model, we saw a moderate improvement in F1 score from the *DeepNet*, as it went up from 0.838 to 0.854 on the dev set. Looking at precision and recall, we see that our precision went down slightly, but our recall also went up, resulting in a higher F1 score. In our *ThreeNet* model, we used most of the same hyperparameters discussed. One important thing to note is that when training, we noticed that our loss remained stagnant while the weights seemed to oscillate. This indicated that we were oscillating around a minimum, and we needed a lower learning rate. We proceeded by incorporating stepwise learning rate decay and after some experimenting, we were able to get the best results with a learning rate of 2e-4, with decay 0.1, using *1/t* decay. In the end, both precision and recall went up from the *TwoNet*, and our overall F1 score also went up from 0.854 to 0.876. Finally, to get conclusive results, we ran our best model of each on our test set and saw the F1 score improve steadily between models, as expected (Figure 4).

Model / Metric	DeepNet	TwoNet	ThreeNet
Train Precision	**0.976161**	0.944275	0.947189
Train Recall	0.734467	0.780178	**0.815891**
Train F1	0.838239	0.854419	**0.876651**
Dev Precision	**0.974962**	0.943015	0.946456
Dev Recall	0.734696	0.780194	**0.814567**
Dev F1	0.837946	0.853912	**0.875573**

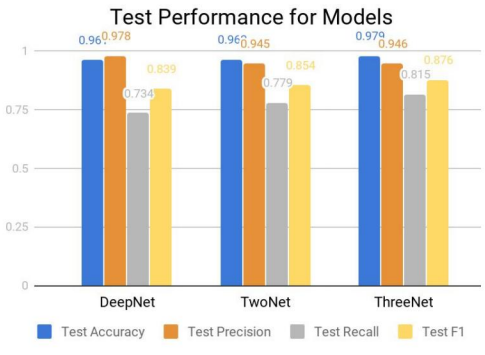


Table 3 (left): *Precision recall and F1 scores across train/dev sets for our models. The model with the best recall and overall F1 score was ThreeNet, as expected.*

Figure 4 (right): *Aaccuracy in blue, precision in orange, recall in gray, F1 score in yellow of our models on the test set.*

While our final models are reasonably accurate, there are specific areas of improvement centered around errors that the models make that could direct our future research. Specifically, when looking at the precision-recall curves for *DeepNet* and *TwoNet*, we

noticed that the two models had highly similar curves, with the *TwoNet* having a slightly larger AUPRC (Figure 5). This indicates that the addition of sequence information to the *TwoNet* provided a small boost in model improvement but was not the paradigm shift we were expecting. In order to improve the accuracy of our *TwoNet* over the *DeepNet*, we could try adjusting the length of the sequence feature; for this project it was fixed at 21 due to the amount of time it took to generate sequence features, but that parameter can be adjusted in the future. One other model architecture we tried was an *EnsembleNet*, created by combining the predictions from the three models. Unfortunately this ensemble was unable to improve the F1 score of our models, suggesting that the three models are making similar errors. The sources of error for our models can be seen in the confusion matrix in Figure 5; while the figure only shows the confusion matrix for the *ThreeNet*, the other figures are similar. As the matrix shows, the main error source is predicting biological errors as human-introduced; in the future we can focus on improving this specific task.

Another visualization we implemented to understand the inner workings of our models was visualizing the activation heatmaps for the hidden layers of our models, as shown in Figure 6. Specifically, we were able to look at how, on average, the models reacted for examples that it correctly predicted, as well as for how the inputs were propagated for examples that it misclassified. Since an ideal model would classify false positives (FPs) as true negatives (TNs), we can use the heatmaps to hone in on features that we should look more deeply into to ensure the activations are similar. We see that the main differences between the activations occur in features 9, 10 and 11: base quality, mapping quality, and fragment length, respectively. After some analysis we saw that the Illumina sequencer is not developed to output very accurate base and mapping qualities, so those features might not be very meaningful. Furthermore, for fragment length, the data we had was not consistent in the lengths of fragments used, as some DNA was sequenced with shorter fragments, while others had larger fragments. From prior research conducted within the Alizadeh Lab, fragment length can be a very important feature. To improve our model, we need to standardize our data and be consistent with how the DNA is sequenced with regards to fragment length, which should help significantly.

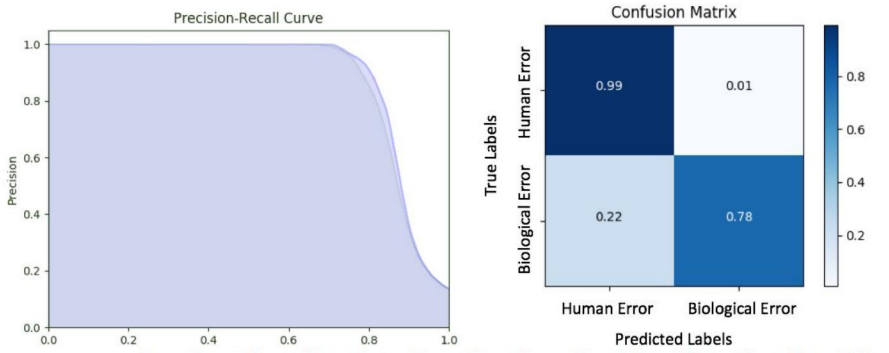


***Figure 5**: Precision-recall curves (left) for DeepNet (green) and TwoNet (blue), and a normalized confusion matrix for ThreeNet (right). The confusion matrix for ThreeNet is highly similar to that of the other developed models, suggesting that a systematic improvement in one could improve all three.*

***Figure 6**: Mean activation heatmaps across the 14 features of the input layer of ThreeNet, with dark blue representing least activated and bright red most activated. An ideal model would classify false positive examples as true negatives, so these respective heatmaps should be similar.*

6. Conclusion & Future Work

This was a challenging project due to the quantity and format of the data as well as the complexity of the models required for this data of this nature. Nevertheless, we were able to make steady improvements in model accuracy and F1 numbers with more complex models, eventually achieving a 97.3% accuracy and 87.6% F1 score for our ThreeNet model. While it may be possible to improve our model through further training and hyperparameter search, which we hope to focus on in our future work, our results so far give us hope that this method can eventually be used in the medical field to improve blood cancer diagnostics.