Using Artificial Neural Networks to Detect Multiple Cancers from a Blood Test

William H. Andress and Steven S. Qu

Dr. Christopher T. Symons

2019

Contents

Abstract

Traditional cancer-screening tests are expensive, invasive, and often limited to a single cancer type. A recently developed blood test, CancerSEEK, had shown promise in simultaneously screening for eight different types of cancers: ovarian, liver, stomach, pancreatic, esophageal, colorectum, lung, and breast, of which the first five types previously had no screening tests. However, CancerSEEKs accuracy was limited by its logistic regression model. Presented here are improved results by using a deep learning artificially intelligent neural network capable of exploiting nonlinear relationships between variables that CancerSEEK could not. The new model, called Cancer Detecting Neural Network (CDNN), was fitted to the same dataset as that used in CancerSEEK for fair evaluation comparison. The dataset was generated from a blood test that detected protein biomarker concentrations and gene mutations from each of 1817 patients. CDNN was then evaluated using ten-fold cross validation, and finally optimized by adjusting hyper-parameters and the output threshold. While keeping the specificity above 99%, the median accuracy was increased from 70.80% in the CancerSEEK model to 82.30% using CDNN.

Acknowledgements

This project would have been impossible without the guidance and expertise of our mentor, Dr. Christopher T. Symons, a Senior Research Staff Member in the Computer Science & Mathematics Division at Oak Ridge National Laboratory. In addition, this work was made possible by the experience and advice of our Math/Science Thesis teachers, Dr. Deanna Pickel and Ms. Jessica Williams, along with the encouragement of our parents.

1 Introduction

Over 17 million people are diagnosed with cancer and over 9 million people die from cancer every year [1]. Cancer is a disease that causes uncontrolled cell division, eventually leading to the development of malignant tumors that can break off and spread cancer to other parts of the body by a process called metastasis [2]. If left untreated, cancer can become lethal. Cancer treatments are most effective when they are used during the early stages of cancer development. For example, the likelihood of someone being alive five years after a diagnoses of Stage I rectal cancer is 88% while the same likelihood for someone diagnosed with Stage IV rectal cancer is 13% [3]. To increase the likelihood of survival, cancer must be detected early, especially before metastasis.

Current cancer screening test methods, such as mammograms, colonoscopies, endoscopies, and cervical cytology, are often expensive, invasive, and limited to a single cancer type [4]. All of these tests rely on physical observations and cannot detect early stage cancer. Blood tests offer a new approach to cancer screening that can overcome these issues. Fragments of cell-free DNA from both cancerous and normal cells are absorbed by the bloodstream. A simple blood test can detect the cell-free DNA while also measuring the levels of various proteins in the bloodstream to diagnose a wide variety of cancer types. Simple machine learning modeling, such as linear regression, of blood tests has been reported, but their accuracy is relatively low due to the complexity of cancer. Machine learning could improve the accuracy of these tests by searching for nonlinear complex relationships between variables. The goal of this project is to create a Cancer Detecting Neural Network (CDNN) by applying machine learning to this data for early detection of cancer with high accuracy and specificity.

2 Background

2.1 Machine Learning

Machine learning is a branch of artificial intelligence based on the idea that machines can learn from data and make decisions on their own [5]. Machine learning uses algorithms and statistical models to improve the performance of a specific task over time. Specifically, machine learning models take in training data to improve the performance of the model before the models are evaluated based on previously unseen testing data. This process is depicted in Figure 1. During the training phase, the model takes in the training data and outputs a prediction. It then compares this prediction with the known output value from the data. Based on this comparison, the model adjusts to better fit the data. After the model is trained, it enters the testing phase. The trained model reads in testing data and produces an output. However, this time, it does not adjust itself; instead, it keeps track of how many correct and incorrect answers it outputs. In classification problems, the output is either binary or multiclass. In binary classification, data points are split into two categories, positive or negative. In multiclass classification, on the other hand, the data can be divided into many different categories.

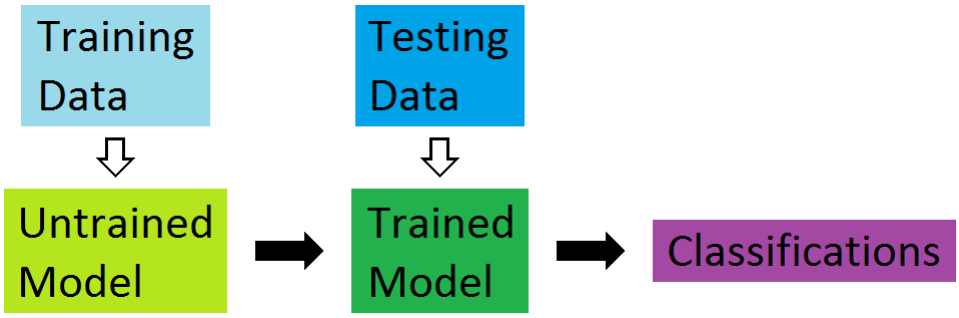

Figure 1: This is a diagram of the training and testing phases of machine learning. Open arrows represent data inputs. Filled arrows represent data outputs.

Shallow learning is a sub-type of machine learning that uses artificial neural networks. Neural networks are computing systems inspired by biological neural networks found in brains. Neural networks can detect nonlinear relationships between parameters, often making them more accurate than other machine learning techniques, such as linear regression [5].

Neural networks work by taking in data and manipulating it to produce a result. The manipulation occurs through the passing of information through different layers of the neural network. There are three different types of layers: input layer, hidden layers, and output layer. While there is only one input and one output layer in each neural network, there can be any number of hidden layers. Figure 2 is an example of a neural network with two hidden layers. Each circle in Figure 2 represents a node; a column of nodes represents a layer. Each layer is made up of at least one node. Each node in a given layer receives data from the previous layer's nodes, manipulates the information, and passes it along to the next layer. The neural network learns to perform tasks from examples in the training phase and is evaluated in the testing phase. In the training phase, the data is inputted into the model many times, and the model is adjusted to produce a more accurate result each time. In the testing phase, the model is left unchanged, and data different from that used in the training phase are inputted to evaluate the model.

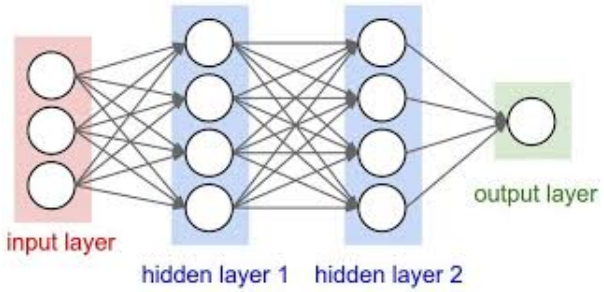

Figure 2: This is an example of a binary classification neural network with two hidden layers [6].

2.1.1 Nodes and Layers

Nodes are storage points for the data as it is being manipulated and transferred. Each arrow in Figure 2 represents the passing of information from one node to another. As this information is passed, it is multiplied by a weight specific to the arrow. Therefore, each node receives a weighted value from each of the previous layer's nodes. The node then adds these values together and stores it before transferring it on to the next node. This transferring of values can be represented by the dot product

$$w_i \bullet \mathbf{a} = b_i \tag{1}$$

where w_i represents the weights corresponding to one node, **a** represents the vector of values of the previous layer, and b_i represents the value of one node in the new layer.

Each node receiving data has a different vector of weights. The transferring of all of the values of one layer to the next is via dot products of different weight vectors by the original layers values. All of these vectors are combined to make a weights matrix to represent the transfer of information of all of the nodes in one layer to all of the nodes in the next by

$$W\mathbf{a} = \mathbf{b} \tag{2}$$

where W represents the matrix with columns corresponding to the weight vectors of each node and **b** is a vector of the values of the new layer's nodes.

The nodes manipulate the data based on activation functions. These activation functions are the same for each node in a layer but can be different across layers. Two such activation functions are the Rectified Linear Unit (ReLU) and sigmoid functions shown in Figure 3. In the ReLU function (Figure 3a), any negative input is replaced with zero while non-negative inputs are left unchanged. The sigmoid function (Figure 3b) scales the input to a probability value between 0 and 1. The ReLU function is the default activation function for hidden layers while the sigmoid function is commonly used in the output layer for binary classification [5].

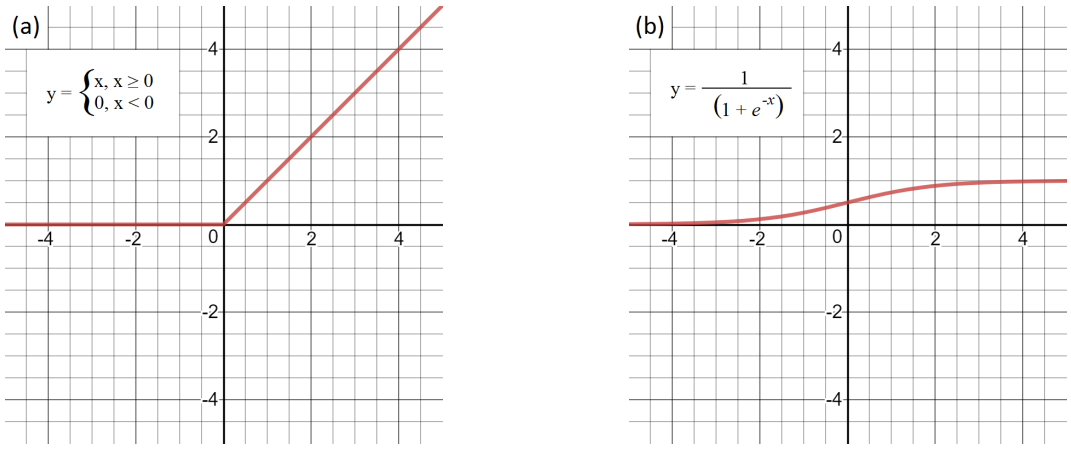

Figure 3: Part (a) shows the equation and graph of the Rectified Linear Unit activation function, and part (b) shows the equation and graph of the sigmoid activation function.

The input layer is composed of an equal number of nodes as the number of parameters, or variables, in the dataset. In Figure 2, the input layer contains three nodes, meaning the original dataset only consisted of three parameters. In the input layer, the data is merely copied, so it can be manipulated without affecting the original data. The output layer has one node in binary classification problems, but it can also have more when doing multiclass classification. The sigmoid activation function in the output layer is used in binary classification to calculate a probability between 0 and 1. This probability is compared to a threshold. If the probability is greater than or equal to the threshold, the network returns 1. Otherwise, it returns 0. This threshold changes the output of the network from a continuous value to a discrete one.

2.2 Underfitting and Overfitting

Underfitting and overfitting are two common issues in machine learning. Figure 4 shows the three possibilities for the fit of a neural network: underfitted, good fit, and overfitted. Underfitting occurs when the network is not complex enough to correctly model the training data as shown in Figure 4(a). Overfitting, on the other hand, refers to an overly complicated network [5] as shown in Figure 4(c). While this network may be a good fit for the training data, it probably will not model the testing data well because it fits itself so tightly to the training data, missing the general trend. Figure 4(b) is a balance between the two where the model closely resembles the general trend and is therefore a good fit.

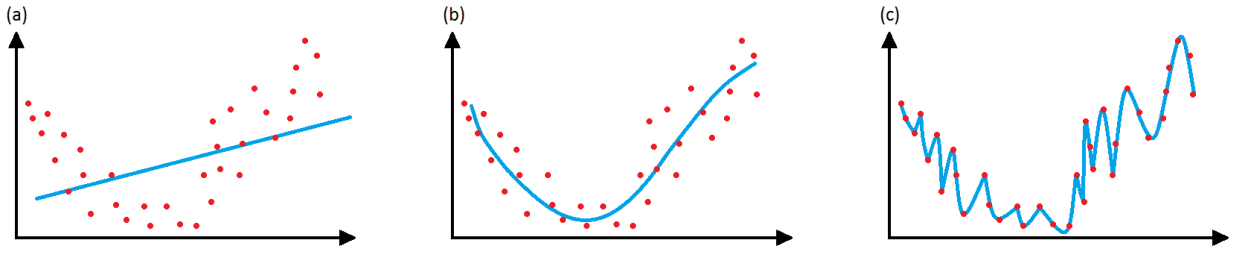

Figure 4: Graph (a) shows an underfitted model, (b) shows a good fit model, and (c) shows an overfitted model where the red dots are the data points and the blue line is the model.

2.2.1 Regularization

A neural network learns to model a dataset by minimizing a cost function, which measures the error of the model [5]. However, during this process, the model can become overly complex, resulting in overfitting. To prevent this, a regularization term is added to the cost function of each layer [5]. The regularization term raises the cost of models with large weights, therefore, making the network treat them as unfavorable. This can help reduce overfitting by making simpler models more favorable than complex ones. The regularization term can be adjusted to vary the cost associated with large weight matrices. For example, it is unfavorable to make the term too large since that can result in the network ignoring smaller, but possibly still important, relationships. There are several different types of regularizers including *L2* regularization, which is given by

$$L2 = \sum_{i=1}^{n} W_i^2 \tag{3}$$

where n is the number of nodes and W_i is the sum of the weights of each layer. The value of *L2* is then multiplied by the regularization weight denoted by λ. Optimizing λ is necessary to prevent underfitting and overfitting. The regularized cost function is given by

$$C' = C + \lambda * L2 \tag{4}$$

where C represents the non-regularized cost function, C' is the regularized cost function, λ is the regularization weight, and *L2* is the regularizer.

2.2.2 Batch-size and Epochs

Often, a dataset is too large for a neural network to analyze in one run, so the dataset is divided into smaller batches. The batch-size refers to the number of training examples in a single batch. When the model is training, the batches of data are run through the network many times, where each run is called an epoch. After each epoch, the neural network adjusts the weight matrices

in order to minimize a cost function and improve the performance. In order to do this, the neural network must be trained with the optimal number of epochs and batch-size to avoid both overfitting and underfitting.

2.2.3 Cross Validation

Cross validation is a method of data splitting [5]. The purpose of this method is to simulate how well the model would perform on unseen data and highlight any possibility of overfitting. The data is split into k partitions. Each partition is then used as testing data once and as training data for the rest of the iterations. Each of these different data divisions is called a fold. For example, a common type of cross validation is 10-fold cross validation. In this type, the complete dataset is split into 10 smaller subsets. The model will then fit itself ten times, each time training on nine-tenths of the data and leaving a different tenth of the data to be used as that fold's testing data. Figure 5 shows the first and sixth folds of cross validation, demonstrating how the training data and testing data is different for each fold.

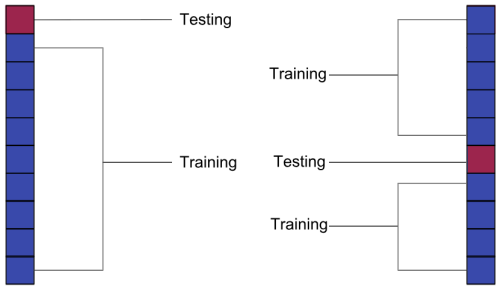

Figure 5: Two folds of ten-fold cross validation where in the first example the first section is used as testing data and in the second, the sixth section.

2.2.4 Hyperparameter Optimization

Collectively, batch-size, epochs, layers, nodes per layer, and regularizers are known as hyperparameters. These hyperparameters can be adjusted to improve the effectiveness of the model while avoiding overfitting. One method of optimizing hyperparameters is called grid search [5]. In this method, each hyperparameter starts with an initial value. Hyperparameters are increased

until the model stops improving significantly, at which point it reverts back to the model with the highest performance. The process is preformed to optimize each hyperparameter. Sometimes it is necessary to run this method multiple times since changing one hyperparameter could affect the optimal value of another. Figure 6 shows the grid search process for optimizing the number of nodes and layers of a neural network. Each row in the figure corresponds to the hidden layer being optimized. The dashed black boxes represents the optimal structure for each layer and the solid black box symbolizes the overall optimal structure. For example, in the first row, the neural network keeps increasing the number of nodes until it reaches four nodes when it determines the accuracy has stopped improving significantly. Therefore, it reverts to the previous structure with three nodes. Similarly, the number of hidden layers follows the same optimization process. Each row in Figure 6 represents the addition of a new layer. The neural network starts with one hidden layer and finds that the accuracy improves after adding a second layer. Therefore, it tries to add a third layer, but this time it determines the accuracy has not improved significantly, and reverts back to the two hidden-layer model, denoted by the solid black box.

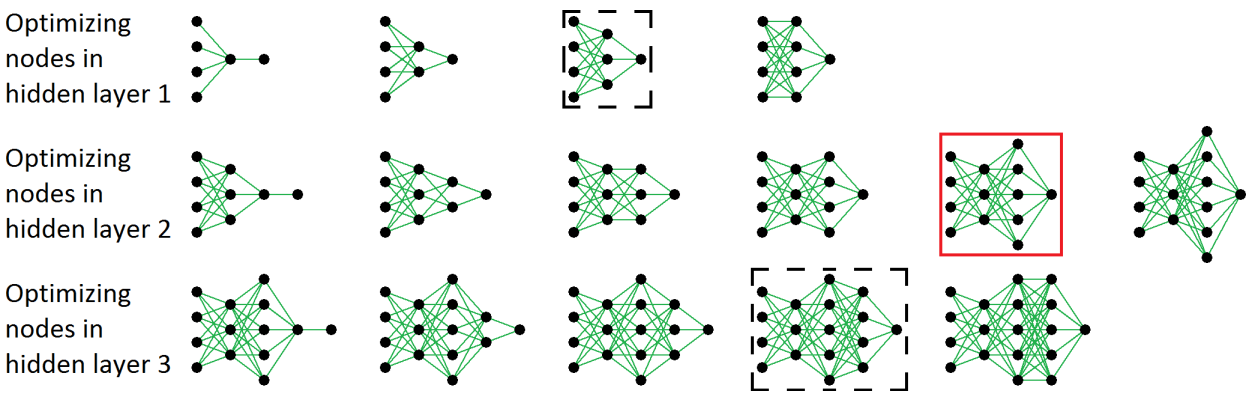

Figure 6: An example of hyperparameter optimization using grid search is shown. The network with the highest accuracy for each layer is outlined by a dashed black box. The network with the overall highest accuracy, and therefore the most optimal structure, is outlined by a solid red box.

2.3 Evaluating the Neural Network

After the neural network is built and trained, the model must be evaluated. Often times the neural network will exhibit high accuracy on the training data because it has been altered to model

that data. However, to simulate the model's true performance, new, unseen data must be used. This phase is called the testing phase. During this phase the model is not adjusted while the testing data is passed through it many times and the performance each time is recorded. The average performance of the neural network is often a good estimate of how the model will perform in practice, on a new, unknown data set.

2.3.1 Sensitivity and Specificity

Specificity and sensitivity (Figure 7) are statistical measurements of a neural network's performance in binary classification tests, such as cancer detection [7]. Sensitivity is the true positive rate while specificity is the true negative rate. Although having high values in both sensitivity and specificity is important, early stage screening tests are designed to prioritize specificity over sensitivity to avoid unnecessary, expensive follow-up tests and mental stress in healthy individuals. In Figure 7, each small circle represents a data point. The filled circles are true positives and the open circles are true negatives. Data points within the large green circle are classified as positive by the neural network and those outside are classified as negative. Sensitivity is the proportion of true positive points that are captured within the green circle to total positives. Specificity is the proportion of true negative points outside the green circle to total negatives.

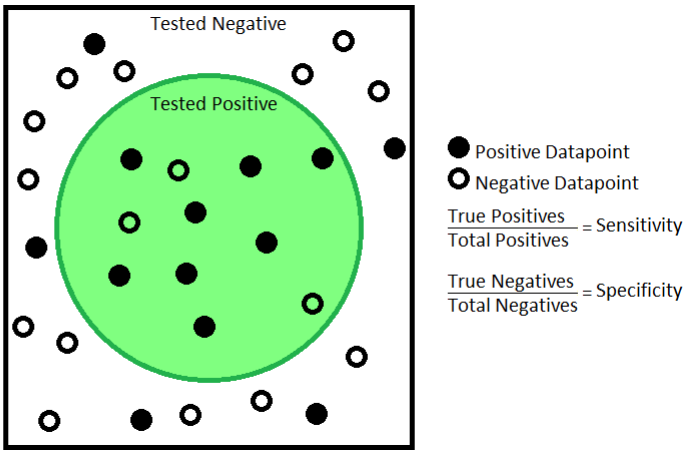

Figure 7: This diagram shows sensitivity and specificity of a binary test where the green region encompasses points classified as positive by the test.

2.3.2 Thresholds

One way to enhance the performance of a neural network is to change the threshold of the output layer. The default value of the threshold is 0.5 meaning that output values of 0.5 or greater are replaced with 1 (positive) while other values are replaced with zero (negative). Changing the threshold value allows the neural network to be customized to fit different scenarios. Therefore, adjusting the threshold is a common technique to alter the sensitivity and specificity of tests. For example, if a test's goal is to prioritize specificity over sensitivity, the threshold of the output layer should be raised. This means that the neural network will not produce a positive result unless it is fairly certain. Increasing the threshold different amounts will vary the neural networks inclination to return a negative result instead of a positive one.

2.3.3 Receiver Operating Characteristic Curve

A Receiver Operating Characteristic (ROC) curve is a parametric graph of true positive rate (sensitivity) and false positive rate (1-specificity) with t being the common parameter [8]. The values for t are different threshold values ranging from zero to one. The ROC curve is monotonic, increasing and bounded by both the x- and y-axes between zero and one. In Figure 8, two example ROC curves are shown. A neural network can be evaluated based on the area under the ROC curve. The larger the area under the curve, the closer the neural network gets to sensitivity and specificity of 100% or perfect accuracy under an optimal threshold. The red curve in Figure 8 is better than the blue curve because the area under the red curve is larger.

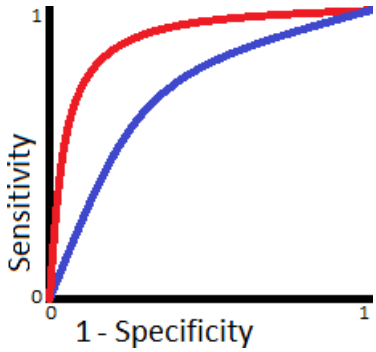

Figure 8: In this ROC Curve, the red curve is preferred to the blue curve because the area under the red curve is larger.

2.4 CancerSEEK

An economical blood test, called CancerSEEK, was developed by a group at Johns Hopkins University that simultaneously detects eight different cancer types based on concentrations of certain protein biomarkers in cell-free DNA [9]. CancerSEEK is able to detect cancer of the ovary, liver, stomach, pancreas, esophagus, colorectum, lung, and breast. Out of these eight types, the first five previously had no screening tests [9]. Since esophagus and stomach cancers require the same follow-up test (endoscopy), they were combined into the common category of upper gastrointestinal (GI). CancerSEEK measured the level of 39 different protein levels and mutated genes in cell free DNA to separate healthy patients from those with cancers. The protein levels were normalized. The mutated genes were used to produce a value called an omega score. The formula for omega score produced a high value for those mutations which were prevalent and a lower score for the mutations which were less common. The omega score converted the discrete mutation values to a continuous score. A simple machine learning technique, logistic (linear) regression, was run on the protein levels and omega score to find linear relationships between those parameters and produce a classification prediction with 70.80% median accuracy over the cancer types and specificity of 99.14%.

3 Methods

Data and result analysis were performed using Microsoft Excel©. Programming for the Cancer Detecting Neural Network (CDNN) construction and adjustment was performed using Python 3, the back-end mathematics library TensorFlow™, and the front-end machine learning library Keras™. Experiments were done on a computer with a 1.8 GHz Intel® Core™ i5 processor and 8 GB of RAM.

3.1 Data Splitting

The same data used to train and test CancerSEEK was used in CDNN to provide a fair comparison between the models. The dataset included information from 1817 individuals, 1005 with non-metastatic clinically detected cancer and 812 healthy individuals. The data for each individual is made up of concentrations from 39 protein bio-markers in the blood and an omega score based on the occurrence of mutated genes in cell-free DNA. The data was divided using ten-fold cross-validation to produce training and testing data from the complete dataset in a ratio of nine to one.

3.2 Initial Model

To begin, a simple neural network was created with an input layer of 40 nodes (39 protein bio-markers plus 1 omega score), one hidden layer of five nodes, and an output layer of one node. The ReLU activation function was used for the hidden layer and the sigmoid function was used for the output layer. *L2* regularization with a regularization weight of 0 was used to keep the first model as smiple as possible for optimization. The model was trained and tested using ten-fold cross validation, and was initially run over 80 epochs with a batch-size of 32.

3.3 Optimizing Hyperparameters with Accuracy

After construction of the original model, the hyperparameters were optimized using the grid search method based on accuracy. For each fold of the cross validation, the number of nodes in the hidden layer was increased by five and the process was repeated with the data from the next fold. This process was repeated until increasing the number of nodes did not improve the accuracy by more than 0.1%. At this point, the number of nodes in the first layer was set to the previous number since it had resulted in the highest accuracy. This process was repeated again to optimize the regularization term and epochs. Given the size of the dataset, the batch-size would not significantly affect the performance of the neural network and was left unchanged. After optimizing one layer, another layer was added to the network and the process of optimization was repeated for this new layer. The entire process was repeated until increasing number of layers did not lead to higher accuracy. At the end of this process, the network was fully optimized. More than ten folds were required to fully optimize the network, so after each tenth fold, the data was redivided for reuse.

3.4 Optimizing Hyperparameters with ROC Curves

Optimizing the network based on accuracy resulted in an approximately equal number of false positives and negatives. However, since the purpose of the test is to screen for early signs of cancer, the network should have a high specificity (low false positive rate). In order to reduce the number of false positives, the threshold of the output layer was increased so the model would classify a patient as positive for cancer only if it was fairly certain in its diagnosis. Therefore, for each fold of cross validation, an ROC plot was generated and the model was optimized based on the area under the curve in addition to the accuracy.

3.5 Adjusting Thresholds

Optimization based on area under the ROC curve produced a model best suited to having its threshold adjusted to raise the specificity while maintaining a high accuracy. This new network was then run on all folds of the dataset during cross validation. Instead of having the output layer round its result, the result was left as the probability of cancer based on the neural network's analysis (a number between 0 and 1). Then the threshold was increased to limit the number of false positives until the specificity rose as close to 100% as possible without significantly decreasing the sensitivity.

4 Results and Discussion

In this section, the development of the Cancer Detecting Neural Network (CDNN) is outlined over each of the optimization methods used.

4.1 Neural Network Optimization Based on Accuracy

In the first step of optimization, the model was adjusted based on its outputted accuracy. Here, CDNN was more accurate than CancerSEEK, raising the sensitivity from 62.29% to 91.24% and the median accuracy from 70.80% to 95.58%. However, it was lacking in a crucial category: specificity. Whereas CancerSEEK performed at a specificity above 99%, CDNN only produced a specificity of 90.76%, meaning that out of every 9 healthy individuals, on average 1 will be falsely diagnosed with cancer. As previously discussed, this is severely undesirable; therefore, the model was further optimized using ROC curves to reduce the number of false positives.

4.2 Neural Network Optimization Based on ROC Curves

Figure 9 shows the two ROC curves with the highest and lowest areas, (a) and (b) respectively, generated from the ten folds of cross validation in the final model. The x-axis shows the false

positive rate and the y-axis shows the true positive rate. In addition, each graph shows the area for that given fold. The equation $y = x$ is shown as a dashed line as a benchmark for comparison.

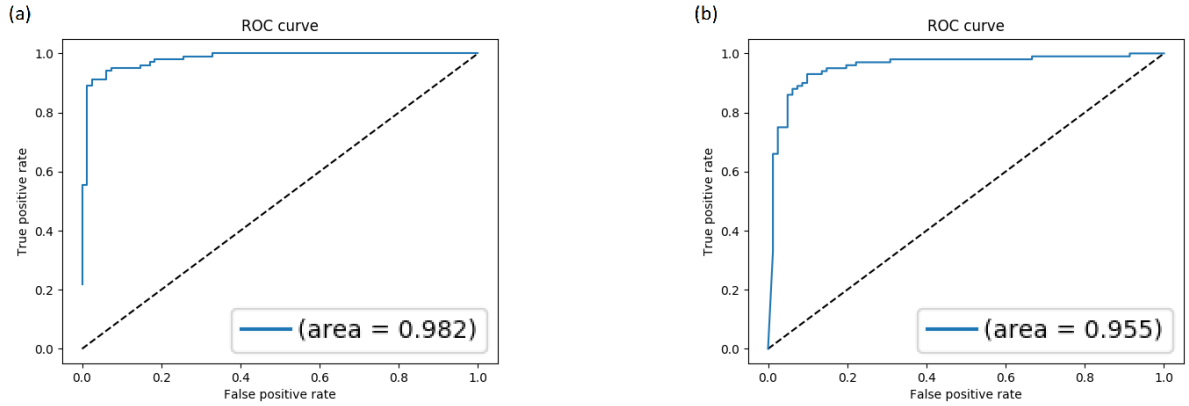

Figure 9: Pictured are the ROC curves (shown in blue) from the two folds of cross validation with the highest and lowest area under the curve. The dashed line is the equation $y = x$ and is used as a benchmark for comparison. Figure (a) had the largest area (0.982) out of the ten-folds, and figure (b) had the smallest area (0.955).

The goal of this optimization was to create a model that generated ROC curves with the largest possible area under the curve. As seen in Figure 9, the maximum area during evaluation was 0.982 and the minimum was 0.955, giving a range of 0.027. This small range is important because it shows the model is able to consistently detect cancer accurately. After optimizing based on area under ROC curves, the final neural network included two hidden layers containing 30 and 25 nodes respectively. In addition, an *L2* regularization with $\lambda = 0.0005$ was added to the second layer to prevent overfitting. The model was run over the data 150 epochs with a batch-size of 32. Finally, the model produced the most favorable results under these conditions at a threshold of 0.96.

4.3 CDNN vs. CancerSEEK

The final version of CDNN had a higher detection rate for all cancer types except ovarian, for which the accuracy remained the same, while maintaining a specificity above 99%. The sensitivity was raised from 62.29% to 74.83% and the median accuracy from 70.80% to 82.30%. The accuracies of the individual cancers ranged from 52.63% for breast cancer to 100.00% for liver

cancer. Table 1 shows side-by-side comparisons of the individual cancer detection rates between CDNN and CancerSEEK.

Table 1: Detection of Individual Cancer Types for CancerSEEK to CDNN

	CDNN Accuracy	CancerSEEK Accuracy	Difference
Lung	83.65%	58.65%	**25.00%**
Breast	52.63%	33.49%	**19.14%**
Upper GI	82.30%	70.80%	**11.50%**
Colorectum	76.29%	64.95%	**11.34%**
Liver	100.00%	97.73%	**2.27%**
Pancreas	74.19%	72.04%	**2.15%**
Ovary	98.15%	98.15%	**0.00%**

CDNN was able to match or exceed the performance of CancerSEEK in the detection of every cancer type with an increase of only one false positive. From these results it is clear to see that some cancer detection rates were improved much more significantly than others. These cancer types were divided into three groups based on increased accuracy: over 15% (green), between 5% and 15% (yellow), and less than 5% (grey). The green group (Lung and Breast) benefited the most from using CDNN over CancerSeek's linear model, implying that few linear relationships were present. The yellow group (Colorectum and Upper GI) showed significant improvement, suggesting the presence of nonlinear relationships. Finally, the grey group (Liver, Ovary, Pancreas) was classified as a relatively insignificant change, meaning that CDNN performed at the same level as CancerSEEK. This implies that the relationships in this group were mostly linear or dependent upon other factors other than protein bio-marker concentrations and gene mutations since CDNN was not able to improve the results significantly. However, it is worth noting that the detection rates of Liver and Ovarian cancers were already highly accurate. Table 2 shows mean accuracy, median detection accuracy, sensitivity, and specificity as well the change of each from CancerSEEK to

CDNN.

Table 2: Overall Comparison of CancerSEEK to CDNN

	CDNN	CancerSEEK	Difference
Mean Accuracy	85.64%	78.76%	**6.88%**
Median Accuracy	82.30%	70.80%	**12.50%**
Sensitivity	74.83%	62.29%	**12.54%**
Specificity	99.01%	99.14%	**- 0.13%**

Table 2 illustrates the overall effectiveness of CDNN versus CancerSEEK. The median accuracy of the cancer types increased by 12.5% while mean accuracy increased 23.35%. These gains and the 12.54% increase in sensitivity came with a loss of only 0.12% specificity, corresponding to one additional false positive. These overall results suggest that there are many nonlinear relationships present in blood data. Analysis of these nonlinear relationships is sufficient to detect cancer. CDNN was able to consistently and significantly outperform CancerSEEK while producing only a small decrease in sensitivity (0.12%).

5 Conclusion and Future Work

Blood tests represent a feasible alternative to traditional cancer screening tests since they are able to accurately and economically identify many cancer types efficiently. To improve upon previous modeling techniques, a four-layered neural network, called CDNN, was built and evaluated using ten-fold cross validation. Specifically, the improvement by cancer type were 33% to 53% for Breast, 65% to 76% for Colorectum, 98% to 100% for Liver, 59% to 84% for Lung, maintaining 98% for Ovary, 72% to 74% for Pancreas, and 71% to 82% for Upper GI. Although there was a slight increase in false positives (CDNN: 8 vs. CancerSEEK: 7), it was deemed insignificant.

More research is required to discover the full potential of artificial intelligence in cancer detection and classification. This project raises the question of whether a more advanced neural network would improve upon the results by detecting more nonlinear connections. In addition, since the relationships between variables varies in linearity depending on the cancer type, creating a separate model for each cancer type may prove to be more accurate at a low computational cost.

While detecting cancer is the singular goal of most screening tests, blood tests must also locate the cancer in the body. Future work on this project is geared towards expanding CDNN with multiclass classification to not only detect the cancer but also determine its location in the body. Potential steps include either generating cancer-specific binary classification neural networks or changing the current neural network into one of multiclass classification.

6 References

[1] Worldwide cancer statistics. https://www.cancerresearchuk.org/health-professional/cancer-statistics/worldwide-cancer. Accessed: 2019-1-12.

[2] What is cancer? https://www.cancer.gov/about-cancer/understanding/what-is-cancer. Accessed: 2019-1-12.

[3] Survival rates for colorectal cancer, by stage. https://www.cancer.org/cancer/colon-rectal-cancer/detection-diagnosis-staging/survival-rates.html. Accessed: 2019-1-12.

[4] Screening tests for cancer: Facts on early cancer detection. https://www.medicinenet.com/cancer_detection/article.htm#cancer_screening_test_facts. Accessed: 2019-1-12.

[5] Francois Chollet. *Deep Learning with Python*. Manning Publications Co, 2018.

[6] Convolutional neural networks. http://cs231n.github.io/convolutional-networks/. Accessed: 2019-1-12.

[7] D G Altman and J M Bland. Statistics notes: Diagnostic tests 1: sensitivity and specificity. *British Medical Journal*, 308:1552, 1994.

[8] Karimollah Haijan-Tilaki. Receiver operating characteristic (roc) curve analysis for medical diagnostic test evaluation. *Caspian Journal of Internal Medicine*, 4(2):627–635, 2013.

[9] Joshua D. Cohen. Detection and localization of surgically resectable cancers with a multi-analyte blood test. *Science*, 359:926–930, 2019.