cs640/Project Report

Data Science Bowl

2017

Alexander Bennett
Anya Gilad
Arpita Vats

Instructors: Prof. Margrit Betke, Ajjen Joshi

Boston University
Department of Computer Science

April 12th, 2017

Abstract

The Data Science Bowl acts as an international competition where people from around the world lend their expertise to solve a current challenge in a variety of fields. In the Kaggle 2017 Data Science Bowl the problem we needed to solve was how to use data science tools and techniques to improve the detection of early signs of lung cancer from CT scans.
In this challenge we look for models that may aid in the early detection of lung cancer nodules in high risk patients. The early detection of lung cancer improves the survival rate of victims of lung cancer. One of our major challenges comes from the problem itself.
To understand the problem from a different perspective we abstract away from the lung cancer problem and consider it much like an attempt to predict which haystacks have needles in them. We possess labeled images of haystacks with needles and those without. Now we must predict the probability that other haystacks have needles given the images we have.
From the information we possess we must develop model/s to predict the probability of lung cancer in patients. The models that we developed included a 3D Convolutional Neural Network, and a Random Forest decision tree model.

Contents

1 Background

In this section of our report we discuss what topics we needed to understand in order to complete the project.

1.1 Computerized Tomography (CT) Scans and Dicom

The data Kaggle gave us included over a thousand low-dose CT images from high-risk patients in a DICOM format. When a doctor takes a CT scan of a patient they produce cross-sectional images of the body, that, when added together produces a three dimensional image. Each of the CT scans contains a number of NxM dimensional slices that represent a specific depth of the patients body.
Each of these scans is represented in DICOM format. DICOM stands for digital imaging and communications in medicine. It is the standard data format for handling, storing, printing, and transmitting information in medical imaging.

1.2 Models

Once we understood the type of data the Kaggle Data Bowl provided, we needed to make a decision on which models to use. We settled on two models whose tutorials Kaggle made available in the kernels section of the competition, a 3-D convolutional neural network, and a random forest.

1.2.1 3D Convolutional Neural Network

A 3D Convolutional Neural Network or 3D CNN is essentially the same as a regular convolutional neural network but in 3 dimensions. We can break down the operations of a regular convolutional neural network into four main operations. The first convolution, the second Non-linearity, the third pooling or sub sampling and the fourth classification. In our aim of understanding 3D CNNs we need to understand these four operations and the terminology that comes with this type of algorithm.

Convolution

Within the convolution step of a CNN the purpose of the convolution is to extract features from the image. When we extract features we are essentially performing a template matching operation, where we slide a smaller image across the base image and take the dot product of one matrix on top of the other. This produces a smaller matrix of values that have been convolved with the smaller matrix, which is called a filter or kernel. The output matrix of this operation is called the feature map. In practice the CNN learns the values of the filters it uses to create the feature map on its own, as designers it is our job to specify the number of filters, filter size, and architecture of the network before training. As a general rule of thumb the more filters we have, the more image features get extracted and better our network is at recognizing patters in images.
The feature map is controlled by three parameters, that we must decide before performing the convolution step. The depth, which corresponds to the number of filters we use for the convolution operation. The stride which is the number of pixels by which our kernel slides over the input matrix. A large stride produces smaller feature maps. The third parameter is whether or not to zero-pad our input image. Sometimes zero-padding can be a convenient tool to add a border around our image, a convolution with zero-padding is called a wide convolution, without zero-padding it is called a narrow convolution. After the convolution operation we introduce a non linearity to our CNN.

Introducing a Non Linearity

A common non-linearity that is used in CNNs is RELU. RELU is an element wise operation that replaces all negative pixel values in the feature by zero. This step of the CNN introduces a non linearity in our data, which is in many cases necessary to model real world data which is mostly non-linear. After the RELU operation we perform the pooling step.

Pooling Step

In spatial pooling we are reducing the size of the feature matrix and only keeping what we define as the most important features. Spatial pooling may define the most important feature in a variety of ways: Max, Average, Sum etc. If we look at an example of max pooling, we define a spatial window whose dimensions are less than the dimensions of the feature matrix. Then similar to the convolution step we define a

stride in which the spatial window slides across the feature matrix. At every position the spatial window finds the maximum of that window and saves that value into a smaller feature matrix.
The pooling step is necessary because it reduces the feature dimensions and makes it more manageable. It also reduces the number of parameters and computations in the network which makes it less likely to overfit the model. After the pooling step there is the fully connected layer operation.

Fully Connected Layer

The fully connected layer takes the outputs of the pooling layer and uses them to classify the data based on the labels available in the training set. Typically the softmax is used as the activation function in these fully connected layers, but we could also use support vector machines to perform the classification.
Once we have all of these operations we then use forward propagation to propagate our data to the output layer. Then calculate the mean squared error of the output layer and use back propagation to calculate the gradients of the error with respect to all the weights in the neural network. Then we use gradient descent to update all the filter values, weights, and parameter values to minimize this error. We continue doing this until this particular network has been successfully trained. Then once it has been trained we run the model on a test set to gather its predictions on the test set. CNNs are popular tools used for image classification in machine learning, and as an popular tool there have been several python libraries developed to help build these types of models. The tool we used to build our 3D CNN was Tensorflow.

Tensorflow

TensorFlow is an open source software library for numerical computation using data flow graphs. The nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional arrays (tensors) communicated between them. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization. The way TensorFlow works is nuanced. It has many attributes that are similar to python but others that are clearly distinct. We shall discuss these attributes from the ground up. TensorFlow handles objects similarly to the way python handles them, names have objects, but objects don't have a name. Now unlike python that evaluates an expression as soon as you define the relationships in the expression, TensorFlow

lets the user define a relationship within an expression and then have it evaluate sometime in the future. When we define a mathematical operation within TensorFlow that operation ends up becoming added as a node in a graph. This is an important concept when it comes to the efficiency of TensorFlow. TensorFlow does all of its computations outside of the Python environment in other high efficiency languages. The way it does this is that as we define the relationships in our TensorFlow graph, we are defining a graph of interacting operations that once evaluated run entirely outside of python.
Once we have defined all the relationships in our TensorFlow graph and want to run, we must initialize the graph. This is done using a call to initialization and session start in TensorFlow that will be outlined in our problem solution.

1.2.2 Random Forest

Random forest algorithm is a machine learning technique that is increasingly being used for image classification and creation of continuous variable such as present tree cover and forest biomass.Random forest is ensemble model which mean that it uses results for many different model to calculate response. In most cases the result from an ensemble model will be better result from any of the individual models.Random forest are grown on convolution neural network for scene categorization, feature form multi-layers of deep convolution neural network are utilized.A feature selection method is proposed to use random forest categorize scenes is a breakthrough performance achieved in computer vision this project we used random forest because its considered good for the images classification we are using, also it's runtime is quiet fast and they are able to deal with unbalanced or missing data.It is one of the most accurate learning algorithm available, for many dataset it produces highly accurate classifier,and it efficiently runs on large databases.

XGBoost

XGBoost stands for eXtreme Gradient Boosting,it is an optimized distributed gradient boosting library designed to be highly efficient,flexible and portable, it has been dominating in applied machine learning and kaggle competition for structured or tabular data.It's being designed for speeding up the performance.Implementation of the model support the features of scikit-learn implementations.The three main form of gradient boosting supported are gradient boosting, stochastic gradient boosting and regularization gradient boosting.XgBoost is open source software available for user under Apache-2 license.The reason we used Xgboost is because of it's execution speed and model performance, generally Xgboost is really fast when compared with other implementation of gradient boosting. XgBoost dominates structured or tabular dataset on classification and regression predictive modeling problem.XgBoost library implements the gradient boosting decision tree algorithm.

2 Approach

This section discusses our approach to the problem. We outline the risks to our approach, discuss the techniques and tools we use, then we go into detail on our problem solution.

2.1 Risk Analysis:

For our 3-D Conv NN we faced the risk of overfitting the training data. This risk comes from the fact that a 3-D Conv NN requires a large data set to make correct classifications. We may reduce the risk of overfitting in a variety of ways. Some methods we may use include: the acquisition of more data to train with, modifications to our model to make it less likely to over-fit, and the addition of noise to the current training set to add variety in our samples.
The problem of overfitting comes from a model that "fits" more than just the general form of the data but the noise within the data as well. This makes the model something of a specialist on a particular dataset but it means that this dataset is the only one it can correctly predict/classify.
There are other risks that we have to account for when using machine learning. Machine learning relies on statistical models to create models that fit data. Statistical models tend to ignore outliers in data. This becomes a problem when certain patients have had special instruments or surgeries that add noise to the data that the model is not capable of handling. A common way to overcome this problem is by adding as much data as possible to train the model.

2.2 Challenges:

- The data set is very small - only 1590 patients.
- The number of features is very small. We have no information on the nodules position nor information beyond the binary classification.

- The processing power required to run the computations of our algorithms is greater than what most portable computers can handle.
- The memory requirements to handle all the data are greater then what the GPU available to us can handle.
- A single GPU for a group of students means that each group is competing for resource space, and unable to experiment in parallel.
- Third parties changing or modifying the GPU, which causes problems in our ability to run the software on the GPU.

2.3 Techniques and Tools:

Tensor flow, 3D Convolutional Neural Network, Kaggle training set, LUNA dataset.

2.4 Problem Solution

This section of the report will discuss the implementation of the different models we used. For each of the models we can break up our implementation to three sections: pre-processing, training, and testing. At each of these phases there were a number of design choices we were required to make. We shall first discuss the 3D CNN model and each of its steps and then move onto the Random Forest model.

2.4.1 3D Convolutional Neural Network Implementation

As we mentioned in the background a 3D CNN is similar to a regular CNN only in three dimensions. We decided to use a 3D CNN because the data that we were given could be viewed in 3-dimensions, and we believed using a regular CNN going scan by scan would provide to many inputs. We based our 3D CNN model and preprocessing off of a kernel that provided a tutorial on how to pre-process and develop a 3D CNN model for this specific Kaggle challenge. The steps we took for pre-processing are outlined in the section below.

Preprocessing Phase I

In order to process the data for use by our model we needed to understand the structure and format of our data. The CT scans were saved in a DICOM format and

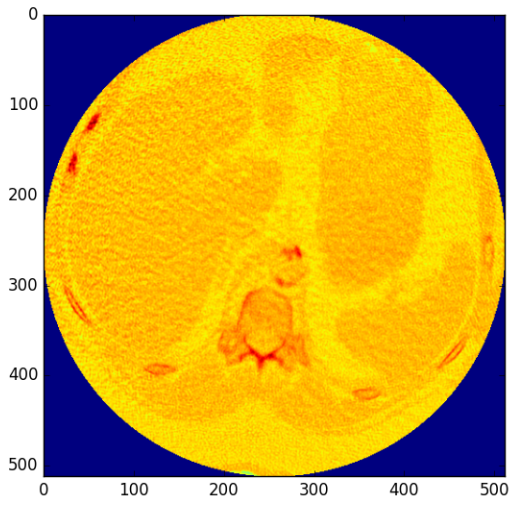

Figure 1: Scan of Lung

were non-uniform in size per sample. Prior to processing these we take a look at one of the samples scans.
Here we see a single scan of the lung. Now that we have an idea of what a single one of these scans looks like we have to look at how many scans are within each sample and decide on a method to chunk these scans so that they are uniform.
Our 3D CNN model required that all inputs should be of the same dimensions, this meant we had to manipulate the data in a way that would standardize the size of the inputs for every sample. We did this by essentially dividing each sample into a discrete number of chunks. For our project we used 20 chunks per scan although more could have been used. We also rescaled our images so that they were 50x50 instead of 512x512. This gave us uniform inputs for every single sample. We tested our 3D CNN on the outputs of this preprocessing phase. We came to the realization that this preprocessing phase did not remove enough of the noise in the samples to train our CNN given the small number of samples. Thus we decided to combine this preprocessing phase with a more intensive preprocessing method.

Preprocessing Phase II

In this preprocessing phase we use the Kaggle Full PreProcessing Tutorial as a reference. The steps in this phase are more intensive than our first phase of preprocessing. In this phase we convert our pixel values to Hounsfield Units (HU), HU are units of

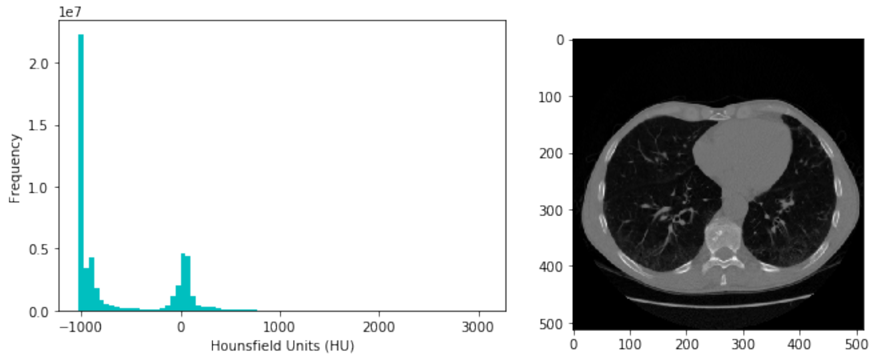

Figure 2: Hu graph and pixel converting

measurement in CT scans and are standardized across all CT scans. Once we convert every scans pixels to HU we standardize the coloring of each scan removing any noise that may be caused by differences in color or brightness. In the figure below we illustrate what a CT scan looks like when it's pixels have been converted to HU.
The next step in this preprocessing phase is to resample the images so that each pixel in a scan represents a specific length, width and height. We resampled each scan to have a 1 by 1 by 1 mm pixel volume. The uniformity in the pixel dimensions will reduce the noise our 3D CNN shall have to deal with because we are making every pixel in every sample this uniform pixel dimension.
After we resampled the image we segment out the lungs. This reduces the information our 3D CNN has to work with and narrows our target zone to help reduce background noise. We included a plot of a three dimensional image of a samples lungs in the figure below.
This lung segmentation fails under specific edge cases that we do not need to worry about in this particular data set.
After we segment out the lungs we then normalize the segmented lung matrix to contain only values between -1000 and 400 in HU. These HU correspond to everything from air to soft tissue, we remove any HU units greater than 400 because they correspond to bones and we are not interested in them.
After we normalize we zero center our final matrix of information. We do this by subtracting the mean value from every element in our array. This centers the mean at 0. Once we accomplished these steps we combined preprocessing phase I with preprocessing phase II to create a dataset that the 3D CNN can use as inputs.

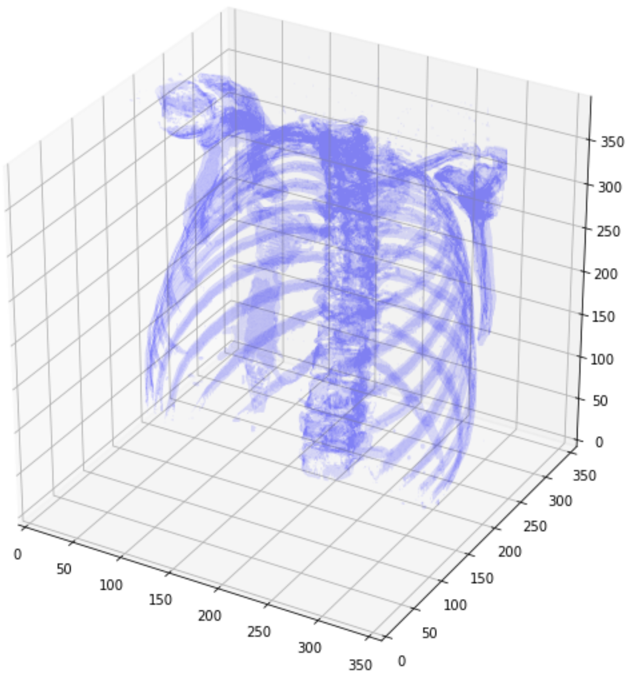

Figure 3: Resampled Image

The 3D CNN Model

We developed our 3D CNN model with the TensorFlow library that was discussed in the background section. There are several functions that our model uses to create the 3D CNN, we shall discuss these functions in detail here.

Function: conv3d

The conv3d function creates a generic 3d CNN with inputs x, weights W and a list of stride values.

Function: maxpool3d

The maxpool3d function defines the type of pooling function we shall be using, and the size of our pooling window. Here we define our pooling window to be a 2x2x2 dimensional cube. We also dictate that our window moves by 2x2x2 strides during every iteration.

Function: convolutional neural network

In the convolutional neural network function we combine all four operations that make up a convolutional neural network into one single function. This includes defining the weights and biases in our two layer CNN model, and defining the non-linearity which

is the Relu function. This function shall be used later on to update the weights and biases of our model.
Function: train neural network
We use the train neural network function to initialize the tensorflow session and perform the forward and backword propagation on our model to train it against the training set. Once we finish training the model we then run the trained model against a validation and test set. The validation set tells us the accuracy of our model, and the output of the model from the test set tells us the predictions the model makes on the test set.
In order to calculate the probability of cancer, we needed to look at the actual output of the model per sample. The prediction of the model per sample is a list with two elements. The 0 element corresponds to no cancer, the 1 element corresponds to yes cancer. Typically we would take the argmax of this list to get a binary prediction, in our case we decided that the best way to calculate the probability would be to take the exponential of each of the values. The exponential handles any cases where any of the values in the list are negative. Then we would sum the two exponentials, we called them x1 and x2. From there we simply did a percentage calculation where we took the part over the whole to find the probability of cancer. We then saved these percentages into a csv file and submitted them to the first stage of the kaggle competition.

3 Experiments

3.1 Preprocessing:

In order to understand the data we are dealing with, we visualized the DICOM images in different variants: 2d images of each slice, 2d images, masks and separating the lungs and the air.
We tried using the size of each image as given and scaled to a particular size.
Our final model uses a fit size of all 3d images.

3.2 Experimenting using the whole data:

1. Before having the stage1 solution, we take 100 patients for validation and 100 patients for testing.

   The validation set was used to compute the loss and accuracy in order to improve the weights, using the Adam optimizer.

   The test set was used to see how well we classified samples compared to the ground truth values.

   Since the data set was very small, we decided to use 20 epochs (after one run with 10, 30 and 40) and see how the log-loss is changing with respect to the different learning rates, as can be seen in figure **??**.

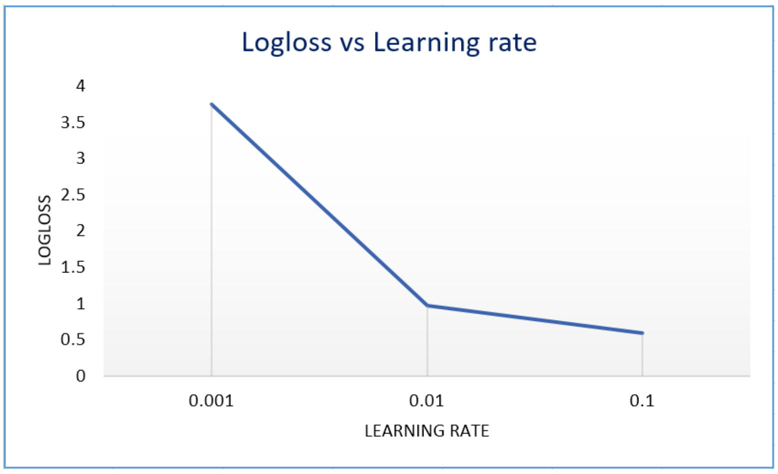

Figure 4: LogLoss vs Learning Rate

2. After getting the stage1 solution - we ran the same conv. net from above, looking at different combinations between learning rate and number of epochs to get the highest accuracy, and the lowest log-loss, using the real stage1 solution.

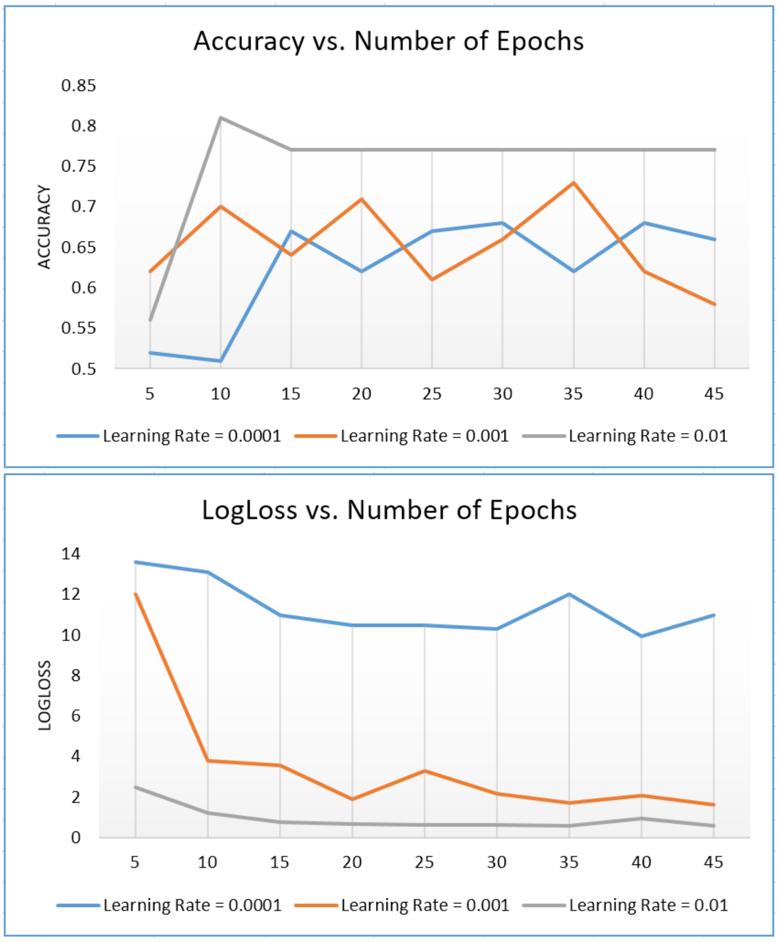

Figure 5: Log Loss and Accuracy vs number of Epochs, for different learning rates

We notice that the lowest log-loss is with learning rate of 0.01, however, by looking at it accuracy of this learning rate, we can see that from 15 epochs and up we get the same accuracy, i.e. the prediction is always the same (in this case, no cancer for all the patients).

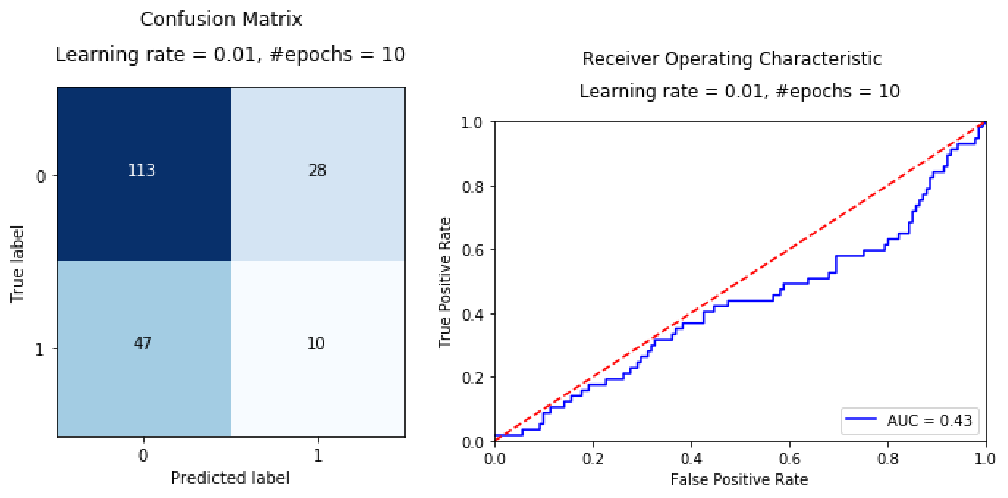

Figure 6: Confusion Matrix and ROC curve

3.3 Splitting the data:

After running the conv. net on the data and checking the log-loss, we noticed overfitting, by seeing that the accuracy is equal to the fraction of the non cancerous patients in the validation set.
Then we tried to apply the same model on 10 subsets of the training set and computed the mean between each models predictions. The LogLoss for this experiment was ≈ 0.62976. The best we got.

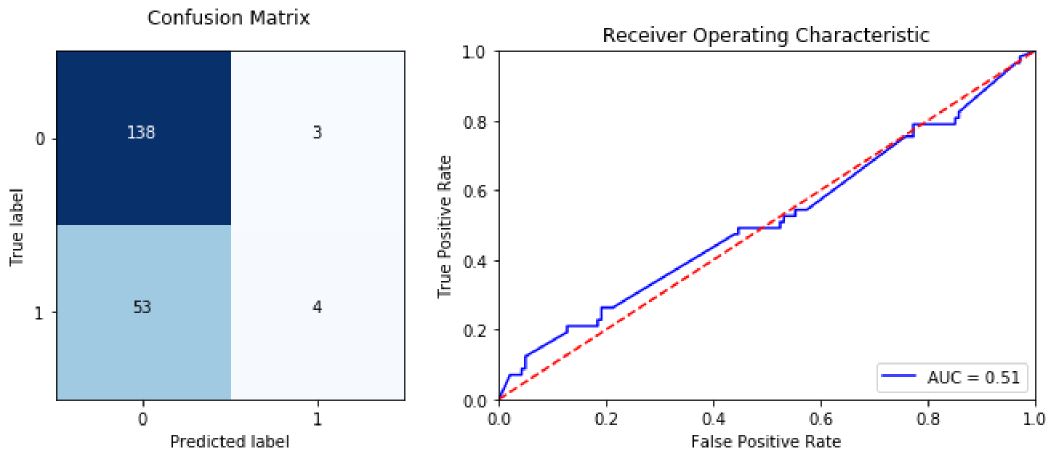

Figure 7: Confusion Matrix and ROC curve

4 Conclusion

4.1 Best solution:

Our best solution to this problem so far is based on solely the phase I preprocessing we take our neural net and train it on 10 different subsets of the full data set. These 10 models become experts on their own specific piece of the data set. We then take the average of all the models prediction on the test set to create a single prediction. We are still running tests using the combination of phase I and phase II preprocessing. We hypothesize that using this combination of preprocessing will improve our accuracy at predicting the test set. We are also conducting other tests involving breaking up the data into different chunks and training multiple models and then using the mean of their predictions to come to a specific answer.

4.2 Future Improvements:

- Experiment with different weight optimization functions - Tensor flow provides 11 different optimizers, which are methods to compute gradients for a loss and apply gradients to variables. For the purpose of this project we used the 'Adam optimizator'. In the future we could see the effects of training our model with other types of "optimizators".
- Experiment with different activation functions.
- Using additional data: 1) Stage2 data. 2) Using the LUNA16 dataset to learn the position of the different cancerous nodules.
- Add additional features to the preprocessing to get a better mask of the lungs.
- Use a learning algorithm to find what the actual answers should have been to give the perfect score. Then use these perfect score answers to tune our model to make the correct predictions. We acknowledge that this method may possibly lead to overfitting for a particular test set.

5 Related Papers and web pages

1. www.tensorflow.org - Covers all the information about the Tensorflow applications.
2. Competitions Kernels:
   - Full Preprocessing Tutorial/Guido ZuidhofFull
   - First pass through Data w/ 3D ConvNet/sentdex
   - Candidate Generation and LUNA16 preprocessing/ArnavJain
3. Random Forest And XGboost:
   - Random Forest:- wgrass.media.osaka-cu.ac.jp/gisideas10/Ned Horning
   - XGBoost Tutorial:- xgboost.readthedocs.io/en/latest/tutorials
   - Introduction to Xgboost:- machinelearningmastery.com/Jason Brownlee
4. Convolutional Neural Networks:
   - Illustrated explanation of CNNs: http://xrds.acm.org/blog/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/
   - Explanation on CNNs: http://cs231n.github.io/convolutional-networks/