Predicting the Presence of Lung Cancer using a Convolutional Neural Network

Adam Pollack, Chainatee Tanakulrungson, Nate Kaiser

Abstract

The use of machine learning to detect, predict, and classify disease has grown exponentially in the past few years, especially for complex tasks such as cancer detection and recognition. During this same time period, convolutional neural networks (CNNs) have exploded in popularity and capability, completely transforming the field of computer vision research. The purpose of this project was to use a CNN to detect cancerous lung nodules in slices of MRI scans and separate them from scans with only normal lung tissue.

Introduction

The objective of this project was to predict the presence of lung cancer given a 40×40 pixel image snippet extracted from a medical image database. This problem is unique and exciting in that it has impactful and direct implications for the future of healthcare, machine learning applications in personal data and decisions, and computer vision in general. The medical field is a likely place for machine learning to thrive, as medical regulations continue to allow increased sharing of anonymized data for the sake of better care. Not only that, but the field is still new enough that our project implements methods at the forefront of technology.

Approach

Due to the complex nature of our task, most machine learning algorithms are not well-posed for this project. There are currently two prominent approaches for machine learning image data: either extract features using conventional computer vision techniques and learn the feature sets, or apply convolution directly using a CNN. In the past few years, however, CNNs have far outpaced traditional computer vision methods for difficult, enigmatic tasks such as cancer detection. Due to time constraints, we used Google's TensorFlow for a robust and efficient implementation of our CNN.

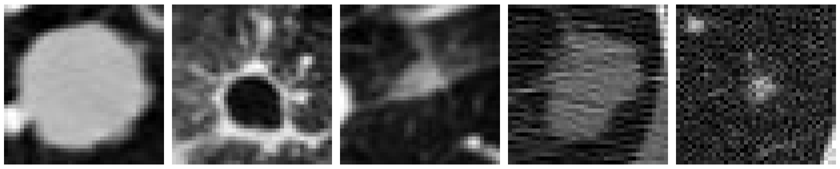

Figure 1 - examples of *cancerous* samples images

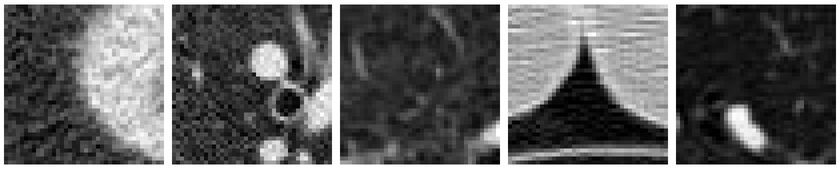

Figure 2 - examples of *non-cancerous* samples images

The images above demonstrate the difficulty of the task. These sample images were pulled from our input dataset. Preprocessing steps to extract these samples from the full 3D lung scans included locating all of the cancer nodules (given a CSV file of groundtruth nodule locations), extracting slices of these nodules for the positive samples, and generating the same number of negative samples by randomly selecting a valid x, y, z coordinate and verifying it was a sufficient distance from the nearest nodule. The full image set was then divided into train, validation, and test sets, and all image pixel values were converted to floating point values between 0 and 1, and zero-mean centered and normalized with respect to the training data.

Tuning a neural network's hyperparameters is a difficult task, even for experienced professionals. We elected to run a large-scale experiment across 5-dimensions for a total of 108[1] test permutations to determine the best hyperparameters. The initial estimates and overall network architecture were based on proven models for similar applications, such as classifying the CIFAR-10 image set.

Table 1: Hyperparameter Permutations

Attribute	**Values Tested**
Kernel Size (1st convolutional layer)	3×3, 5×5, 7×7
Kernel Size (2nd convolutional layer)	3×3, 5×5, 7×7
Number of Filters (1st convolutional layer)	16, 32
Number of Filters (2nd convolutional layer)	32, 64
Dropout Rate	0.1, 0.2, 0.3

Training and evaluating 108 models requires significant computational resources. To complete this project on time, we leveraged resources available through Northwestern's EECS department. Namely, remote access to a Linux server with an NVIDIA GK110 Tesla K20 GPU and access to 19 more Linux machines with NVIDIA GeForce GTX 680s. Training on any one of these machines was ~10x faster than a high-end laptop, and training on all 20 allowed us to vary such a broad range of parameters while still finishing on time.

After evaluating the results from the 108 models with varying hyperparameters, we chose the model that had the highest peak average validation accuracy. That is, the highest validation accuracy before validation accuracy started to decrease and loss started to increase. This model had kernel sizes of 3x3 and 7x7 as well as 16 and 64 filters for the first and second convolutional layers respectively. It also had a dropout rate of 0.2. Using this model, we trained five different network architectures to determine the architecture that gave us the highest accuracy on our test dataset.

[1] 108 = 3 × 3 × 2 × 2 × 3 total variables (3 for kernel size 1, 3 for kernel size 2, etc.)

Table 2: Network Architectures

Network Name	Network Description
Architecture 1	Conv1 -> Pool1 -> Conv2 -> Pool2 -> Dropout -> Dense
Architecture 2	Conv1 -> Pool1 -> Dropout -> Dense
Architecture 3	Conv1 -> Pool1 -> Conv2 -> Pool2 -> Conv3 -> Pool3 -> Dropout -> Dense
Architecture 4	Conv1 -> Pool1 -> Conv2 -> Pool2 -> Conv3 -> Pool3 -> Conv4 -> Pool4 -> Dropout -> Dense
Architecture 5	Conv1 -> Conv2 -> Pool1 -> Conv3 -> Pool2 -> Dropout -> Dense
Architecture 6	Conv1 -> Conv2 -> Conv3 -> Pool1 -> Dropout -> Dense

These various architectures were determined based on the principles described in the Stanford CS231n course notes[###]. All pooling layers had a 2x2 kernel and a stride of 2. The first convolutional layer had a 3x3 kernel and 16 filters, the second had a 5x5 kernel and 32 filters, the third had a 7x7 kernel and 64 filters, and the fourth had a 7x7 kernel and 64 filters. All convolutional layers were followed by a rectified linear unit (ReLU) activation function.

Each model was trained on 2,064 images (batch size of 104), validation was run every 10 epochs on another 442 images, and a final test was run after 500 epochs on another 442 images. After running the final six architectures, we found the inflection point of the loss to be around 250 epochs. We then ran the same setup on each of the six architectures for 250 epochs and recorded the final test accuracy.

Results

After training 108 hyper parameter configurations on our convolutional neural network and then training six different network architectures of one of the best configurations, our best solution (Architecture ##) attained a test accuracy of 96.38%. We ran further analysis to extract a confusion matrix and misclassified images of the final test results to determine why this number was not closer to 100%.

Confusion matrix:

	Predicted +	Predicted -
Actual +	226	12
Actual -	4	200

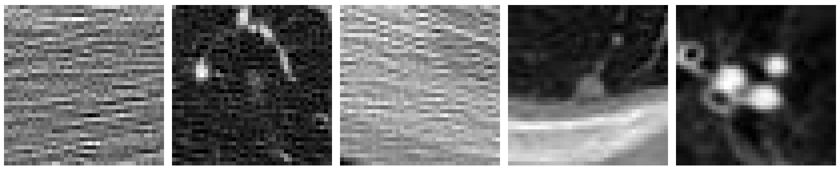

Figure 3 - examples of misclassified test images

Our model classified more examples as negative when they should have been positive than vice versa. We believe this is because of the nature of some of the positive examples. For example, the first four misclassified images above are all positive examples of cancer even though two of them have almost no distinct features. It is likely that it would be just as difficult for a human to classify those images as a doctor. We also can't guarantee that the data we used is completely correctly classified; it is possible there are some mislabeled images.