Recurrent Network for Lung Cancer Classification

Beibei Du Cole Springate-Combs Hongyan Wang (contributed to milestone)

Abstract

We participated in the Kaggle lung cancer detection challenge. The aim is to use chest CT scans to predict whether or not a patient has lung cancer. We tested a simple tree network that takes advantage of transfer learning, and tried to improve upon this by using RNN across the scan slices to allow the network to learn 3D information and to manage the multiple instance nature of detecting any nodules in hundreds of scans. Our results demonstrate some effectiveness of the simple tree network, but our RNN strategies offered no improvement.

1. Introduction

Lung cancer affects almost a quarter of a million people each year in the United States alone. Early detection can help to improve rates of recovery and survival. Currently, lung cancer is diagnosed with the help of chest scans. For each patient, a lung scan can generate between a few dozen and up to a few hundred cross-sectional 2D images of the chest. A radiologist must examine all of the images, noting shape changes between adjacent slides and identifying lung nodules - then deciding whether each nodule is cancerous or not. This is a lengthy process with potential for error; automated systems are in use but better ones are needed.

With the aim of improving state of the art automated cancer detection, Kaggle is running a competition where participants use thousands of lung scans to develop an algorithm that can determine when lesions in the lung are cancerous. They are especially looking to reduce the false positive rate— a problem in current technology— and to free up radiologist time.

There are numerous systems in existence that use machine learning methods for the task of identifying cancerous nodules. Many of the methods use a two-step process: the first step is a region generating method, which identifies candidate regions that might contain pulmonary nodules, and the second step is to use these regions in a classifier that determines whether the nodule is cancerous [8,9,4]. This process has the benefit of not only detecting cancer, but also identifying which region it might be in.

To address limited data, time, and resources, networks ideally take advantage of transfer learning. While lung cancer scans seem highly specialized, there have been successful projects that uses pre-trained CNN's and fine tuned them to detect lung cancer nodules. For example, Ramaswamy and Truong demonstrated the efficacy of transfer learning from general image classification: they used AlexNet and GoogLeNet pre-trained on Imagenet, and by fine tuning, were able to achieve good scores on predicting whether a nodule was cancerous [6].

More pertinent to this project, Fierro et al. showed that by using the deep residual network ResNet (pre-trained on Imagenet) as the feature generator, they were able to train a boosted tree that achieved reasonable results at predicting whether the complete set of a patient's scans contained any cancer (with no specific nodule detection) [3].

One problem with these models is that they often lack the ability to take advantage of the 3D information across sections, (for example Fierro et al. simply average across sections). While there are 3D CNN's that have been used on chest scans[4], Ypsilantis, Petros-Pavlos, and Montana achieved good results with a model that added RNN to a traditional 2D CNN [5]. They used their network on patches that might contain cancer nodules (the size of the recurrence was always seven slices, so this created a sort of fixed size voxel).

The aim of this competition is to identify the probability that a given set of scans contains any cancer, there is no need to specifically identify regions. We attempted to take advantage of the success of transfer learning, and add RNN LSTM features that will allow the model to learn 3D information, deal with the multiple instance nature of nodule detection, and also work with the varying number of slices per patient.

Our experiment used whole images (no patches or nodule suggestion generation), and we used RNN over all slices, with one output. We looked at how adding the LSTM layers changed performance vs the boosted tree model previously described.

We used transfer learning with a CNN pre-trained on ImageNet as a featurizer to generate features from the dataset. We used the stage 1 data provided by the Kaggle Data Science Bowl 2017. Two state-of-the-art pre-trained CNN models, ResNet50 and VGG19 from keras with Tensor-flow backend are used as featurizers. For image classification tasks, the first few layers of a CNN represent lower level features and later layers represent high level features which is specific for image classification. In our model, we extract features from penultimate layer outputs using Keras predict function.

Once features are computed, these features are fed into our classifiers to train a good classifier. Our preliminary experiment is to use a boosted tree LightGBM to do image classification on the validation dataset. Performances are compared based on Log Loss and the ROC curve.

2. Background

2.1. VGG [1]

The VGG network architecture was introduced by Simonyan and Zisserman in their 2014 paper. This network is characterized by its simplicity, using only 3x3 convolutional layers stacked on top of each other in increasing depth, reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier. Architecture of VGG19 is shown in Figure 1, we can see that input image size is 224x224x3 and output a vector of 1000 which corresponds to 1000 classes of ImageNet. Table 1 shows weight layers of VGG19. This network secured first and second places in the localization and classification tracks respectively. They use ImageNet datasets, but also its representation generalizes well in other datasets and can achieve state-of-the-art results, so we use it as one of our pre-trained model to extract features.

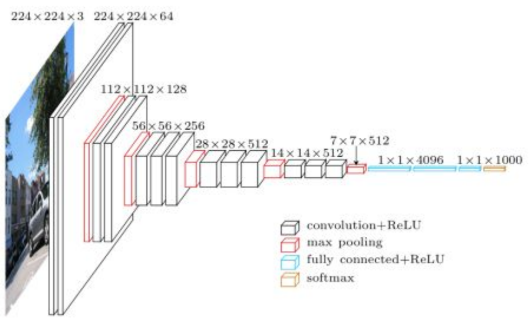

Figure 1. Visualization of VGG19 architecture

Input (224 x 224 RGB image)
Conv3-64 x 2
Conv3-128 x 2
Maxpool
Conv3-256 x 4
Maxpool
Conv3-512 x 4
Maxpool
Conv3-512 x 4
Maxpool
FC-4096 x 2
FC-1000 x 1
Soft-max

Table 1 VGG19 weight layers

2.2. ResNet[2]

ResNet was first introduced by He et al. in their 2015 paper, unlike other traditional sequential networka like VGG, ResNet is instead of a form of 'exotic architecture' that relies on micro-architecture modules. ResNet architectures were demonstrated with 50, 101 and even 152 layers, the deeper ResNet got, the more its performance grew. ResNet won ILSVRC 2015 with an error rate of 3.6%.

The basic idea behind a residual block is to let input x go through conv-relu-conv series to get F(x), then that result is added to the original input x which call that H(x), so H(x) = F(x) + x. Traditional H(x) would just be equal to x. Figure 2 shows one mini module which is computing a 'delta' or a slight change to the original input x to get a slightly altered representation.

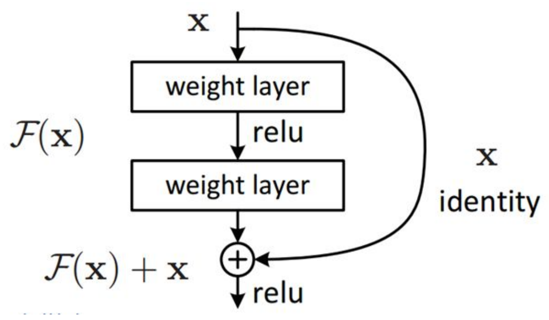
x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2. Residual Network Block

We use ResNet50 in this paper as our other pre-trained model, with input image size 224 x 224 and a depth of 3, and outputs a vector of 1000 probabilities from the final layer, corresponding to the 1000 classes of ImageNet. Table 2 shows weight layers of ResNet50.

Layer Name	Output size	Layers	Number of layers
conv1	112 x 112	7 x 7 x 64, stride 2 3 x 3 maxpool, stride 2	1
Conv2.x	56 x 56	1 x 1, 64 3 x 3, 64 1 x 1, 256	3
Conv3.x	28 x 28	1 x 1, 128 3 x 3, 128 1 x 1, 512	4
Conv4.x	14 x 14	1 x 1, 256 3 x 3, 256 1 x 1, 1024	6
Conv5.x	7 x 7	1 x 1, 512 3 x 3, 512 1 x 1, 2048	3
	1 x 1	Average pool, 1000-d, softmax	

Table 2. Weight layers of ResNet50

3. Approaches

In this section we describe various approaches that we used to work on detecting and classifying lung cancer.

3.1. LightGBM on top of ResNet50 Features and VGG19 features [3]

This part will give a detailed explanation for our model using LightGBM on top of ResNet50 and VGG19 features. We will talk about the basic architecture to do image classification, the method we used to generate features from pre-trained models, the process to train a boosted tree LightGBM and how the predictions are computed for the validation dataset.

3.1.1 Basic model Architecture

A lot of deep learning applications use pre-trained models as a basis, and then apply it to new domain with a related problem, which is called transfer learning. For image classification, the first few layers of a convolutional neural networks usually represent low level features, while the last few layers represent high level features, particularly for specific image classification tasks. In this paper, we use two pre-trained models as our featurizers. The model architecture is similar, we will throw out the last layer of pre-trained CNN models, feed outputs from penultimate layers into the model predict function to generate features, then a boosted tree using Microsoft LightGBM is applied to classify the image.

Figure 3 represents a ResNet CNN with an image from ImageNet. The input image has a size 224x224 and depth of 3, corresponds to three color channels (RGB). The image is exposed to convolutions in each internal layer, diminishing in size and growing in depth. The last layer outputs a vector of 1000 classes of ImageNet and the predicted class of network is the component with the higher probability.

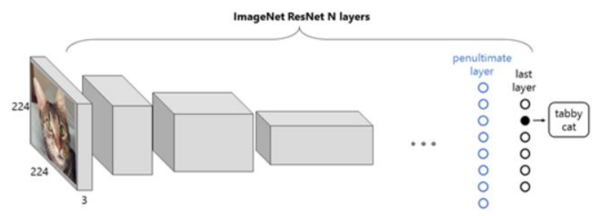

Figure 3: Representation of ResNet [3]

3.1.2 Extract Features

Pre-trained models such as VGG, ResNet in Keras applications are made available alongside pre-trained weights, weights are downloaded automatically when instantiating a model, in this paper, we use these two pre-trained models VGG19 and ResNet50 to do feature extraction. Figure 4 shows our proposed model. Images/Slices of a patient scan are cropped to 224x224 and packed in groups of 3, in order to match format of the ImageNet (which has three channels for color), then these images are fed into the neural networks in batches and convoluted in each internal layer until the penultimate layer, outputs from penultimate layers are transformed into features via Keras model feature extraction function. Note that the generated features shape is different for different patients since every patient scan includes a different number of slice. Also VGG19 has feature shape (k,7,7,512) and ResNet50 has feature shape (k,1,1,2048), k represents number of batches into neural networks relate to every patient, assume a patient scan has n slices, then k = n / 3 (images are packed in group of 3). Finally, these features are set as the input of the boosted tree, programmed with LightGBM which classifiers the images as those of a patient with or without cancer. Figure 4 shows the workflow of proposed solution, models VGG19 and ResNet50 use this same workflow.

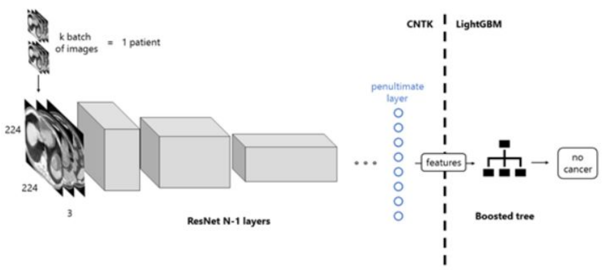

Figure 4. workflow of the proposed solution [3]

3.1.3 Train LightGBM classifier

LightGBM is a gradient boosting framework which uses tree based learning algorithms. Features generated from the pre-trained models are fed into the LightGBM classifier. We use two approaches to train LightGBM classifiers. Approach 1 is to calculate average features for every patient id, then uses these average features as trained data, approach 2 is to use every image features for every patient to be the trained data (so image image gets the ground truth value of the patient). We do cross-validation with a train to validation rate of 8:2 to train LightGBM classifiers. We also tested combining the ResNet and VGG features. Detailed comparison for these two approaches is shown in Table 3, which shows approach 2 takes much longer time than approach 1 for both ResNet and VGG, but with lower final log loss. In the experiment section, we will compare performance on validation dataset using the trained classifiers by these two approaches.

ResNet50	Approach 1	Appoach 2
Train Data Shape	(1397, 2048)	(81942, 2048)
Train Time (seconds)	577.780	5299.740
Stopping Rounds	3164	5000
Final Train Log Loss	0.435	0.414

VGG19	Approach 1	Appoach 2
Train Data Shape	(1397, 25088)	(81942, 25088)
Train Time (seconds)	6419.99	46979.59
Stopping Rounds	2890	5000
Final Train Log Loss	0.4337	0.4026

Combine Two	
Train Data Shape	(1397, 27316)
Train Time (seconds)	6574.270
Stopping Rounds	3553
Final Train Log Loss	0.436

Table 3. Comparison of two approaches

3.1.4 Using LightGBM to do image classification

During the training process, we have two approaches to train LightGBM classifiers. Similarly, we also have two approaches to for the validation. For approach 1, we use mean features of the input data to train the classifier, so we also compute predictions using mean features of the validation dataset. For approach 2, we train the classifier using features of every image of every patient, so we will compute predictions for every image of every patient in the validation dataset, which means every slice will have a probability, then to make it more reasonable, we will get average probability, maximum probability, minimum probability for all images of every patient as the probability of if this patient has cancer and also compare performance based on these terms which is shown in results section.

3.2 CNN and LSTM

We built numerous new networks on top of features generated with the pre-trained models. We used CNTK since it allows for efficient computation of RNN across series of different sizes. Since we will not be identifying potential nodules, our network needs to learn to predict that a patient has cancer based on the existence of at least one node. The idea was that using recurrence with RNN LSTM layers would allow the network to learn the 3D dependencies between slices, and also manage the fact that cancerous nodules only appear on some slices, but we want an output per patient. The use of RNN also allows for a dynamic 'depth', so we will not need to down or upsample the data into a consistent size. This is especially useful since the number of slices per patient varies substantially.

3.2.1 Model

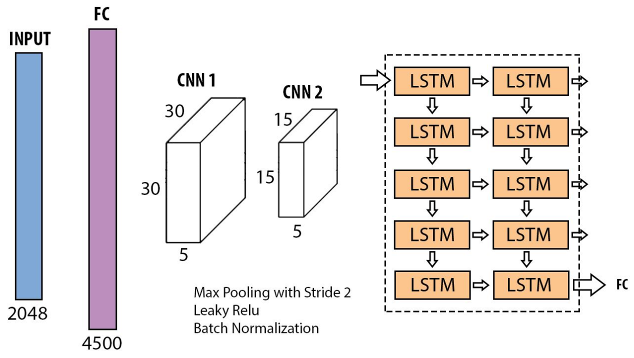

Diagram 1. RNN model overview

The basic model can be seen in diagram 1. The input is first connected to a fully connected layer, and then to two convolutional layers (each with a max pooling of stride 2), and then this is fed into the RNN block. LSTM was chosen because it has been known to reduce the vanishing gradient problem, and should be better equipped than alternatives to deal with the large number of slices. The fully connected and convolutional layers are followed by Leaky Relu and batch normalization.

The RNN block consists of two LSTM layers, each of size 100. The output from the block is fully connected to our final prediction layer, which undergoes softmax.

We used Adam for optimization with a learning rate of .001 and a momentum of 0.9, and weights were initialized with Glorot normal [10, 11]. Loss was computed using LogLoss on the final softmax output.

3.2.2 Variations

Several variations were tested on this basic model. We first tried using only the last layer of output from the RNN block, since this has seen all of the data. We also tried taking the mean and max of all of the LSTM block outputs, and we tested the performance of removing the CNN layers and the input was fed straight into the RNN block.

3.2.3 High Mean

We tested a novel layer that serves a similar purpose as taking the mean or max of the RNN outputs, but should be better suited for our multiple instance problem. Since cancerous nodules are only going to be present in some of the slices, ideally the mean should not be brought down by slices that do not have cancer if there are some that do. So the new layer takes the mean of layers that are above the mean of all of that patient's layers; the idea being these are the slices most likely to contain cancer. This can be thought of as a weighted average where all of the slices that were below the mean are weighted with zero (and not counted in the denominator).

Computationally, it is possible to achieve this with two RNN layers, one that computes the mean and one that works backwards and computes the mean of those layers above the mean. This made it possible to integrate into the CNTK framework.

We tried two versions, one that reduced the mean of layers to a single number and used those for comparison, and one that took the mean across the slices axis, and then selected layers where more than 50% of the matrix was higher than the mean matrix.

4. Experiments and Results

4.1. DataSet

In this study, we used the stage 1 CT imaging data provided by the National Cancer Institute of Kaggle 2017 Lung Cancer competition.

Training data is large, about 140G. In this dataset, it includes 285380 CT images from 1595 patients, every patient has an patient id and every id includes a set of 2D slices, the number of slices is different for different patients with interval [94,541]. Every slice has the same shape of (512,512). Figure 5 shows the distribution of the number of slices per patient. We can see that most patients have about 150 CT images.

Based on the data we have, we need to resize the data set in order to feed them into our model. We use two pre-trained model VGG19 and ResNet50, the input shape of image of (224,224,3), so we need to resize data into (224,224), then group three of them to get shape(224,224,3) and then feed them into our model.

The data includes labels for each patient, 1 represents cancer and 0 represents no cancer. Validation data is also from Kaggle, which includes CT images from 198 patients. The portion of cancerous patients is the same in the training and validation data.

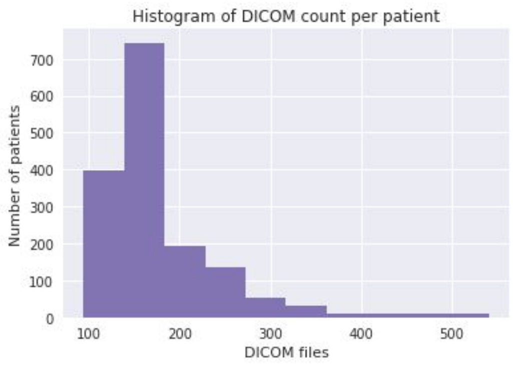

Figure 5. distribution of slices count per patient

4.2. Evaluation metrics

4.2.1 Evaluation using Log Loss

In the Kaggle competition, they use Log Loss as the evaluation metric, so in our experiment, we also compare our performance following this criteria.

$$\text{LogLoss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right],$$

where

- n is the number of patients in the test set
- \\$\hat{y}_i$ is the predicted probability of the image belonging to a patient with cancer
- \\y_i is 1 if the diagnosis is cancer, 0 otherwise
- \\$log($ \\) is the natural *basee* logarithm

Note: the actual submitted predicted probabilities are replaced with \\ $max(min(p, 1-10^{-15}$,10^{-15})\\). A smaller log loss is better.

4.2.2 Evaluation using ROC curve

We also use the ROC curve as our metric to evaluate the performance of different approaches. It's created by plotting TPR(true positive rate) against FPR(false positive rate) at various threshold settings. The primary metric is the sensitivity, or true positive rate, since in lung cancer detection a method that fails to identify will put the patient at risk, another metric is the false positive rate that we hope to reduce it. Figure 6 shows how to analyze performance using ROC curves, the yellow line is best and blue line is worst.

Note that it is possible to achieve a loss of about 0.6 on our data set by predicting that no patient has cancer. However this would result in a worthless ROC curve. For this reason we focused more heavily on the ROC curve results, because they seem better suited to actually understanding the utility of the predictions our models are making.

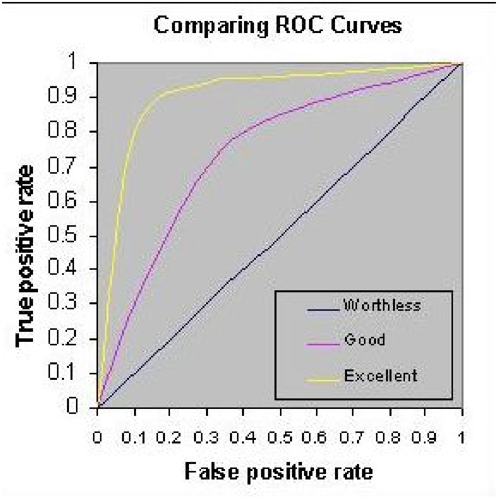

Figure 6. Comparing ROC curves

4.3. Results

4.3.1 GBM results

Table 4 shows log loss on the training and validation dataset using LightGBM on top of two pre-trained models VGG19 and ResNet50 generated features. Comparing results between VGG19_1 and VGG19_2, ResNet50_1 and ResNet50_2, we can find that approach 2 has a lower log loss on train dataset for both models. As we mentioned before, approach 2 means to train LightGBM using every image features generated from train dataset, which means we have more data to train a better classifier, of course, it also takes longer time. We expect approach 2 can have a better performance when computing prediction on validation data, but surprisingly, we can see that approach 1 has lower log loss among all the results for both models. (In order to make approach 2 more reasonable, we get average probability, maximum probability, minimum probability for every patient to find a better one). We also combine the two models mean image features to do training and validation, the result shows that it's no better than training and testing separately.

Model		Log Loss	
		Train dataset	Val dataset
VGG19_1		0.434	0.557
VGG19_2	Avg	0.403	0.582
	Max		0.773
	Min		0.609

ResNet50_1		0.435	0.576
ResNet50_2	Avg	0.414	0.585
	Max		0.617
	Min		0.623
Combine Two model		0.436	0.560

Table 4. Log Loss Comparison
VGG19_1 means when training the classifier, use average features for every patient scan, VGG19_2 means when training the classifier, use all features from every image of every patient scan, same for ResNet50_1 and ResNet50_2

Figre 7 shows the ROC curve using LightGBM on top of Vgg19 features. From left to right, top to bottom, first one is ROC curve via approach 1, second to to last is ROC curve via approach 2.

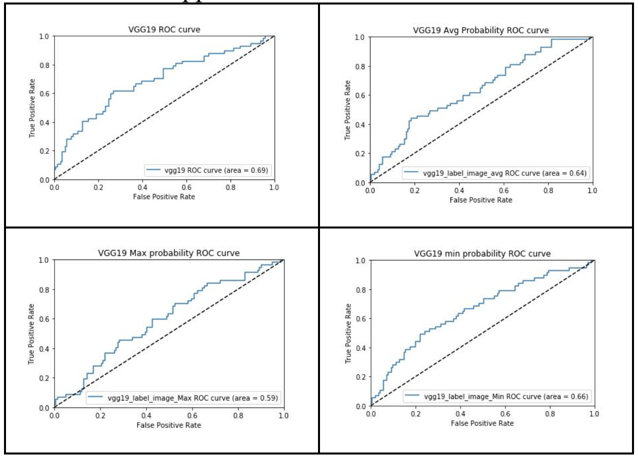

Figure 7. VGG19 ROC curve

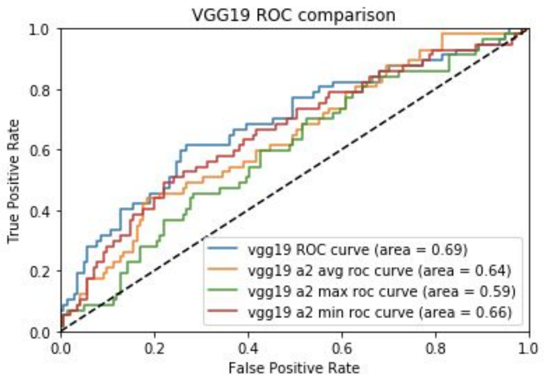

Figure 8. VGG19 ROC curve comparison

Figure 8 shows ROC curve using LightGBM on top of Vgg19 generated features. Comparing areas under ROC curve between approach 1 and approach 2, we can find that approach 1 has a larger area than approach 2, which is compatible with log loss, that approach 1 has a lower log loss. For approach 2, the area under features maximum probability roc curve is smaller than other two areas, and combine with log loss, we also find that it should have a bigger log loss, but areas under average probability and minimum probability has a little conflict with unknown reason, figure 6 shows that average probability has a bigger area but with smaller log loss compared to minimum probability log loss.

Figure 9 shows ROC curve using LightGBM on top of ResNet50 features. From left to right, top to bottom, first one is ROC curve via approach 1, second to to last is ROC curve via approach 2.

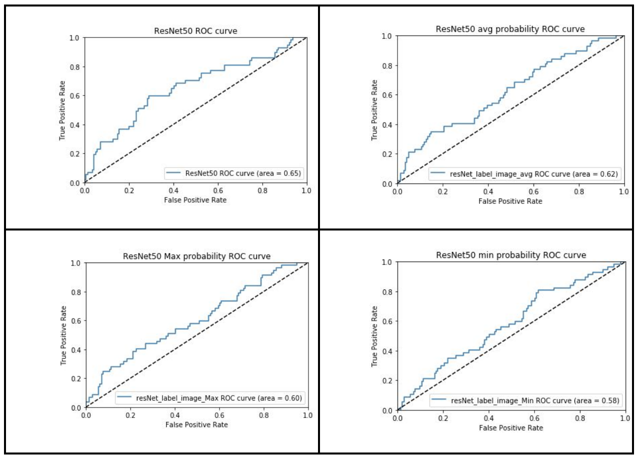

Figure 9. ResNet50 ROC curve

Figure 10 shows ROC curve using LightGBM on top of ResNet50 generated features. Comparing areas under ROC curve between approach 1 and approach 2, we find that approach 1 has a larger area than approach 2, which is compatible with log loss, that approach 1 has a lower log loss on validation dataset which can be found in table 4. For approach 2, area under features minimum probability roc curve is smaller than other two areas, and combine with log loss, we also find that it should have a bigger log loss, which can be verified in table 4. Unlike VGG19 ROC curve, areas under average probability and maximum probability is compatible with Log loss. We can find that area under average probability is bigger than area under maximum probability, and its log loss is smaller.

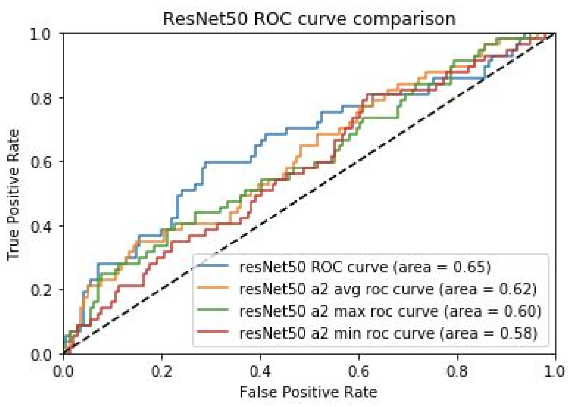

Figure 10. ResNet50 ROC curve comparison

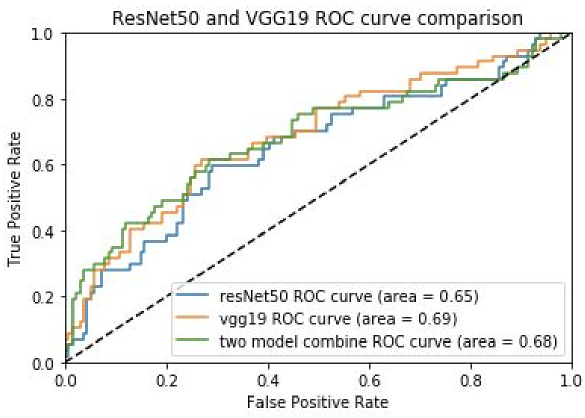

Figure 11. Comparison between ResNet50 and Vgg19 on their biggest area under ROC curve

As Figure 11 shows, we find VGG19 has a bigger area than ResNet50 and two model combine under ROC curve, which is also compatible with their log loss on validation dataset shows on Table 4.

4.3.2 CNN/RNN Results

Networks were all trained on 50000 samples, and they all reached very low training loss (less than 0.05). However this was the result of overfitting, as can be seen in the validation results. We tried adding dropout, limiting training time, and starving the network of parameters, but none of these changes significantly improved our validation results. The full CNN/RNN network was trained with ResNet, VGG, and combined features, and performance was not significantly different, so all variations were tested using only the ResNet generated features to improve training time.

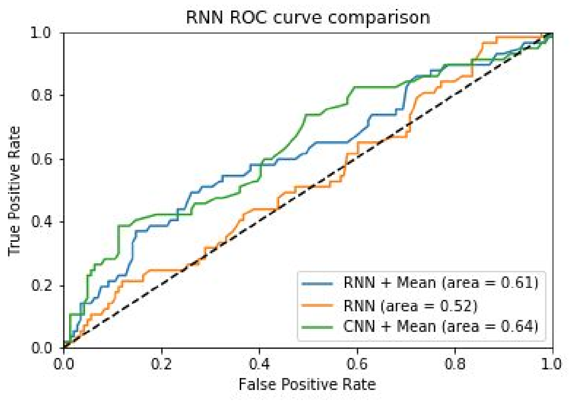

Figure 12. RNN ROC curve comparison

Figure 12 shows the performance of simply adding RNN directly on top of the features. We see that using the last slice of the RNN does not work well at all (close to guessing), but that taking the mean of the RNN outputs works to a certain degree. For comparison, the performance of taking the mean of only the convolutional layers (no RNN) is included. We see that adding RNN in fact seems to offer no improvement.

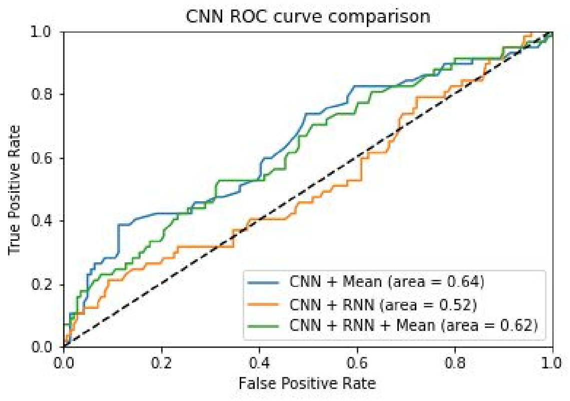

Figure 13. CNN ROC curve comparison

Figure 13 shows the performance of the model variations that include convolutional layers. Again we see that taking the last slice of the RNN block does not work, and that adding RNN and taking the mean works about as well as simply taking the mean after the convolutional layers.

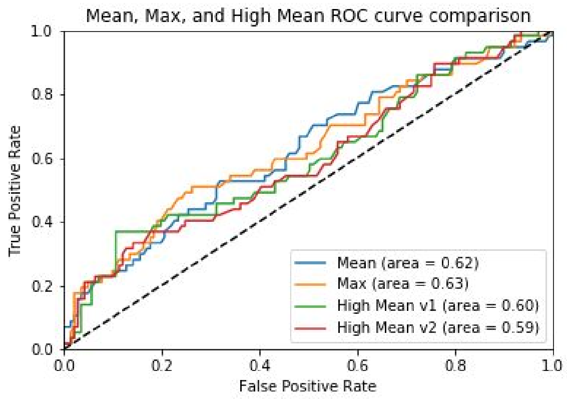

Figure 14. High mean layer vs max, mean

Figure 14 shows the performance of our new high mean layer in comparison to using a traditional maximum or mean. Given that taking the mean worked better than not, we had expected that using the high mean might improve performance, because it should keep the mean from being dragged down by inputs from before the LSTM blocks had a chance to detect cancer. However, we see that while the high mean blocks are better than no mean at all, they perform no better than traditional mean and max.

Model	LOSS	AUC
cnn + mean	0.97	0.64
cnn + rnn	1.44	0.52
cnn + rnn + mean	1.01	0.623
cnn + rnn + highmean v2	9.49	0.59
cnn + rnn + highmean v1	9.94	0.6
cnn + rnn + max	1.03	0.63
rnn + mean	1.21	0.61
rnn	1.33	0.53
data combo, cnn+rnn+mean	1.42	0.57

Table 5. Performance of cnn/rnn networks

As we see in table 5, none of our CNN/RNN models improved upon the performance of our baseline boosted tree experiments. Some came close in terms of AUC, but even then they produced a worse loss.

4.4 Conclusion

We found that transfer learning worked up to a point. Using the boosted tree on top of features generated on pretrained networks, we were able to create a working network, but performance was far from state of the art (CT scans are very different than imagenet images). However the true test was to see if we could improve upon this, using the same features.

We added the RNN blocks with the aim of having the network learn spatial knowledge of cancerous nodes, and also to learn that there only needs to be some slices with cancer to predict that the entire patient has cancer. We were unsuccessful in this; while LSTM blocks can find spatial dependencies in nodules [5], perhaps it was not possible for our setup to learn these dependencies and also fill the multiple instance learning role. Another factor might have been the length of the sequences, which has been known to cause trouble for RNNs.

It was also very easy to overfit the data; training with more sample would have helped. The best submissions to this Kaggle competition also trained on other publically available data. Our network also trained without any separate mechanism to identify potentially cancerous nodes, which more successful models often use.

The high mean layer as it stands is susceptible to skewed distributions, so perhaps a layer that took the top x% most activated slices per patients might perform better (and this x could be a hyper parameter).

If the data had been resampled into a fixed number of slices, we could have put some fully connected layers after the LSTM block (like Ypsilantis et. al.), and this might have improved performance.

References

[1] Karen Simonyan & Andrew Zisserman,Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv:1409.1556 [cs.CV], 2014

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun, Deep Residual Learning for Image Recognition, arXiv preprint arXiv:1512.03385, 2015

[3] Fierro, Miguel, Ye Xing, and Tao Wu. "Quick-Start Guide to the Data Science Bowl Lung Cancer Detection Challenge, Using Deep Learning, Microsoft Cognitive Toolkit and Azure GPU VMs." Blog post. Cortana Intelligence and Machine Learning Blog. Microsoft, 17 Feb. 2017. Web. 10 Mar. 2017.

[4] Anirudh, Rushil, et al. "Lung nodule detection using 3D convolutional neural networks trained on weakly labeled data." SPIE Medical Imaging. International Society for Optics and Photonics, 2016.

[5] Ypsilantis, Petros-Pavlos, and Giovanni Montana. "Recurrent Convolutional Networks for Pulmonary Nodule Detection in CT Imaging." arXiv preprint arXiv:1609.09143 (2016).

[6] Ramaswamy and Truong. “Pulmonary Nodule Classification with Convolutional Neural Networks." (2016).

[7] Yang, He, Hengyong Yu, and Ge Wang. "Deep Learning for the Classification of Lung Nodules." arXiv preprint arXiv:1611.06651 (2016).

[8] Shen, Wei, et al. "Multi-scale convolutional neural networks for lung nodule classification." International Conference on Information Processing in Medical Imaging. Springer

International Publishing, 2015.

[9] Mansoor, Awais, et al. "Segmentation and image analysis of abnormal lungs at CT: current approaches, challenges, and future trends." RadioGraphics 35.4 (2015): 1056-1076.

[10] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[11] Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." Aistats. Vol. 9. 2010.