Exploring 3D Convolutional Neural Networks for Lung Cancer Detection in CT Volumes

Shubhang Desai
Stanford University
shubhang@cs.stanford.edu

Abstract

We apply various deep architectures to the task of classifying CT scans as containing cancer or not containing cancer. We employ a two-stage approach which consists of segmentation of the CT scan into nodule and non-nodule regions using a U-Net architecture, and classification of the regions using 3D CNN architectures. We are able to achieve very good results on the dev set using deep architectures, and have a path for development to optimize further and get better tests results.

1 Introduction

1 in 14 men and 1 in 17 women develop lung cancer in their lifetime, and one of the primary ways to help diagnose the disease is through CT scans. However, finding lung nodules which may be indicative of cancer is difficult to do in CT scans, and being able to distinguish between benign and malignant nodules often requires additional tests such as biopsies. This provides a motivation to create an automated system to aid in the process.

In this project, we explore the use of 3D Convolutional Neural Networks (CNNs) for lung cancer detection in CT scans. Inspired by recent literature, we create a two-stage classification pipeline which consists of 1) segmentation of CT scans to find candidate nodule regions and 2) classification of most-likely candidate regions as cancerous or not-cancerous. We make use of deep architectures for both stages of the pipeline, and use CNNs for 3D convolutions in the second stage. This multi-stage approach is similar to those of recent deep learning papers tackling this problem, but differ in the fact that we attempt to make use of very deep architectures for the classifier.

2 Related Work

Before the onset of deep learning, many papers [1] [15] which attempt to detect lung nodules rely on geometric models to distinguish between regions which are or are not likely to be nodules. This would, of course, need to be pruned to reduce false positive rate. This models saw decent success, but relied on domain knowledge, required extensive preprocessing, and could not generalize to other types of abnormal growths.

Deep models for this task make use of the fact that most successful deep learning applications do not require much feature engineering, and can instead be trained simply on raw input features (in the case of vision, raw pixel values). These papers [3] [16] [9] are able to achieve very good results using custom 3D CNN architectures which are not very deep. We want to see if using very deep architectures (i.e. an 18-layer ResNet [6] and a 121-layer DenseNet [7]) will give us any gains in performance. We decided to try deep architectures as a recent paper was able to detect various ailments in the lungs through analysis of chest X-Rays using DenseNet to produce state-of-the-art results [13].

3 Data

Our data comes from two sources: the LUNA16 We preprocess the CT scans of both the LUNA16 (Lung Nodule Analysis 2016) dataset [10] and Kaggle dataset [4]. The first contains 3D CT volumes and corresponding x-, y-,

and z-locations of nodules in the scans, as well as their radii. Notice that the nodules may or may not be cancerous, and the dataset does not tell us whether the nodules are benign or malignant. The Kaggle dataset contains CT volumes and corresponding binary labels as to whether or not the scan contains cancer. The CT scans from both datasets consist of 512×512 slices, but contain a variable amount of slices (i.e. the z-axis has variable height) depending on the resolution of the CT scanner. Since we are using a 3D convolutional network, each scan itself is an example (about 1000 training examples, 300 validation examples, and 50 test examples)

We preprocess scans from both datasets using a preprocessing kernel on Kaggle, which can be found at [11]. The preprocessing done was to simply mask all regions in the scans that are not lungs. This is done by converting the pixel values of the scans into Hounsfield Units (HU), a measure of radiodensity, and masking all regions in the scan whose HU measures are not consistent with that of lung tissue. We use the raw pixels of the masked CT scans as the input features to both segmentation and classification steps. See part a) of Fig. 1 for an example of data after preprocessing.

4 Methods

Our final detection pipeline consists of two stages: segmentation and classification. We first make use of the LUNA16 dataset, which has both CT scans as well as ground truth coordinates and radii of nodules within each scan. This dataset is used to train a U-Net [14] (an architecture that's popular for biomedical segmentation) to segment 2D scans into segmented predictions of possible nodules. This involves passing a 2D slice into the network, which does a series of convolutions and upconvolutions to encode and subsequently decode an image. Our implementation of UNet as based on open-source source code at [8]. Since our labels are binary, we simply use a sigmoid activation on the last layer of the network, stretch out the output from the network and the ground truth into a vector, and use binary cross-entropy as the objective function to train our segmentation network:

$$l(y, \hat{y}) = \frac{1}{N} \sum_i w_1 y_i \log(\hat{y}_i) + w_0 (1 - y_i) \log(1 - \hat{y}_i)$$

This objective motivates the network to output predictions close to 1 when the ground truth is positive, and 0 when the ground truth is negative. Note that this is the formulation for weighted cross-entropy, as we have a w_c term which scales the loss value; we have a fixed value of w_c for each class $0, 1$ This previous step comes up with a lot of false positives. So, inspired by the work of Chon et al. [3], we take the top 8 regions of size $32 \times 32 \times 32$ and concatenate them into a $64 \times 64 \times 64$ tensor which is used as input to the next stage. One difference between our approach and that of Chon et al. is that we use L1 of the activations over the window to determine candidacy score, instead of L2. We choose this because we can efficiently compute L1 using a 3D convolution with a kernel of all 1's. Note that we take $32 \times 32 \times 32$ slices from the original input scan, not of the segmentation output, as the input to the classification stage. We do not allow the final regions of candidacy to overlap, so once we pick a region, we 0 out the region and all regions which overlap with it to prevent overlapping windows. This is to make sure that regions close to each other which have high candidacy due to the same nodule are not picked multiple times.

The second stage of our pipeline is classification. We make use of the Kaggle Data Science Bowl 2017 dataset, which contains CT scans and corresponding binary labels which describe whether or not the scan contains cancer. We pass each of these scans through the U-Net and get the regions of highest candidacy. Then, we pass the most-likely regions through a variety of baselines and models to achieve as high accuracy as possible. The baselines we use are Naive Bayes and SVM. The models we use are ResNet and DenseNet, modified to take 3D input. For the deep models, we use a series of convolutions to transform the image into a binary prediction. We again use weighted binary cross-entropy as the objective function to train the networks (our weights are different between this training regime and the previous one). We use 3D ResNet and DenseNet architectures as implemented in the open-source repo [12].

Since the inputs for both the LUNA16 and Kaggle datasets come from the same distribution (lung CT scans), we did not believe that there would be an issue with training the segmentation stage with one dataset and the classification stage with another.

All of our code was implemented in PyTorch [2].

5 Experiments

We carry out a series of experiments to train a U-Net for segmentation and various models for binary classification of cancer.

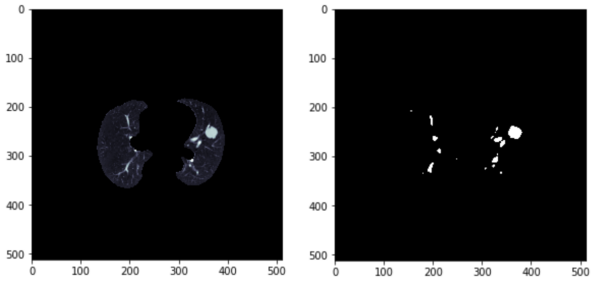
5.1 Segmentation

We train a U-Net to be able to segment CT scans into nodule and non-nodule regions. This is done simply on 2D images, so we do this operation at each slice of the full CT volume over the z-axis. These segments are the agglomerated back into a volume. We then select the top 8 regions of candidacy by evaluating the L1 score over a $32 \times 32 \times 32$ window and concatenating the regions into a $64 \times 64 \times 64$ image.

Shown in Fig. 1 is an example of a segmented slice. After doing a series of hyperparameter-tuning experiments, we find that the most important hyperparameter is the positive class weight. This is because there is much more non-nodule regions than nodule regions in the CT, so even if the network outputs all 0s, it is fairly accurate. So, we set the positive class weight equal to the inverse of the proportion of nodule regions to total volume in the entire training set, which we find empirically to be about 5000. Further experiments find that a learning rate of 1e-2, a batch size of 10, and an Adam optimizer produce the best results. Our trained U-Net has a recall of 0.6073 and a precision of 0.001212. Note that we expect the precision to be quite low, as we have a very large false positive rate. As described above, we prune false positives by taking the 8 regions of highest L1 value as the "most likely candidate regions", and train the classifier on these regions.

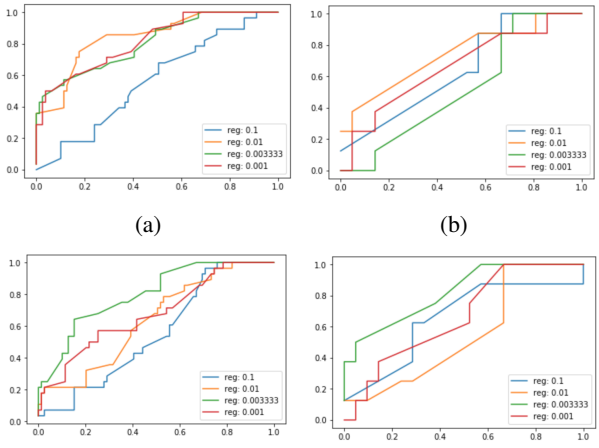
5.2 Classification

We try a number of experimental setups to turn our top segmentation regions into a prediction of cancer. To be able to do experiments quickly, we take a subset of our train, dev, and test sets (1/10th of each original set) to do experimentation on. We try training naive methods such as Naive Bayes and SVM as baseline, and then optimize ResNet and DenseNet hyperparameters to achieve the best possible accuracy. Since we are focused on making sure we have low amount of both false positives and false negatives, we choose our accuracy measure to be the area under the ROC curve. State-of-the-art AUC achieve on the task is about 0.83 [5].



(a) CT input (b) Segmented output

Figure 1: Shown are a) a masked CT slice which is inputed into the segmentation network and b) a segmented output image generated by the network.

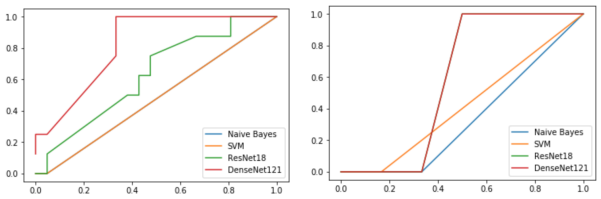


(a) (b)

(c) (d)

Figure 2: ROC curves (plotting precision vs. recall for various thresholds) for a) ResNet on the train set, b) ResNet on the dev set, c) DenseNet on the train set, and d) DenseNet on the dev set. We vary the regularization coffecient to be 0.1, 0.01, 0.003333, and 0.001, respectively.

Shown in Fig. 2 are the ROC curves of a few ResNet and DenseNet experiments on the train and dev sets, as well as their hyperparameter settings. We find that the regularization coefficient is the most important hypeparameter, as the network is quite prone to overfitting. We fix all other



(a) (b)

Figure 3: ROC curves (plotting precision vs. recall for various thresholds) for all models on a) the dev set and b) the train set. Note that in the first graph, NB and SVM have the same curve; and in the second graph, ResNet and DenseNet have the same curve.

Model	Dev AUC	Test AUC
Naive Bayes	0.4762	0.3333
SVM	0.4762	0.4167
ResNet18	0.6280	0.5833
DenseNet121	0.8214	0.5833

Table 1: AUC of the ROC curves for our final models on the dev and test sets.

hyperparemters to: learning rate of 1e-3, batch size of 20, and Adam optimizer. We see that a regularization coefficient of 0.01 for ResNet and 0.003333 for DenseNet give the best results of both the train and dev sets.

We also train Naive Bayes and SVM models on the data to as benchmarks. Shown in Fig. 3 is the comparison of the ROC of our final models (benchmarks and deep models with best hyperparameters) on the dev and test sets. Table 1 shows the AUC of these models.

We see that our deep models are able to significantly outperform our benchmarks on both the dev and tests sets. We are able to get good results on the dev set of the classification task; in fact, we are able to achieve dev AUC which is close to the 0.83 state-of-the-art. However, we are underperforming on our test set. It seems that we have overfit to the dev set. This is likely because we use a small subset of the full training set for experimentation and fast implementation; training on the full set of data would likely allow the network to generalize better.

6 Conclusion

Our models are able to get very good results on the dev set. However, there seems to be overfitting occurring on the train/dev sets, so some further steps would be: 1) train a better segmentation stage, as the current recall and precision do not reflect great performance; 2) tune hyperparameters to get better generalization on classification; and 3) use the full dataset instead of a small subset of the data to allow for better generalization on classification. We believe that completing the third step would be the most beneficial to test performance, as our train, dev, and test sets are small enough that training may be overfitting on the train and dev sets.

However, we have demonstrated as a proof-of-concept that it is possible to get better results on the task of lung cancer detection by using deeper models. We are excited by this finding and where we may take it.

Contribution

Since I was the only person on the project, I was responsible for all of the project's contributions. I received guidance and advice from Awni Hannun at Stanford Artificial Intelligence Lab (SAIL) throughout the project.

Acknowledgements

I would like to thank the instructors and TAs of CS229 for an informative quarter. I would also like to thank Awni Hannun for his continuous guidance and support throughout this project.

References

[1] Junjie Bai, Xiaojie Huang, Shubao Liu, Qi Song, and Roshni Bhagaliaa. Learning orientation invariant contextual features for nodule detection in lung ct scans. 2015.

[2] Soumith Chintala and Adam Lerer. pytorch. `https://github.com/pytorch/pytorch`, 2015.

[3] Albert Chon, Niranjan Balachandar, and Peter Lu. Deep convolutional neural networks for lung cancer detection. 2017.

[4] Booz Allen Hamilton and Kaggle. Data science bowl 2017, 2017. data retrieved from LUNA16 site, https://www.kaggle.com/c/data-science-bowl-2017/data.

[5] S. Hawkins et al. Predicting malignant nodules from screening ct scans. 2016.

[6] Kiaming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 2015.

[7] Gao Huang, Zhuang Liu, Kilian Q. Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. 2016.

[8] Jackson Huang. unet-pytorch. https://github.com/jaxony/unet-pytorch, 2017.

[9] Xiaojie Huang, Junjie Shan, and Vivek Vaidya. Lung nodule detection in ct using 3d convolutional neural networks. 2017.

[10] Colin Jacobs, Arnaud Arindra Adiyoso Setio, Alberto Traverso, and Bram van Ginneken. Lung nodule analysis 2016, 2016. data retrieved from LUNA16 site, https://luna16.grand-challenge.org/data/.

[11] Arnav Jain. Candidate generation and luna16 preprocessing. https://www.kaggle.com/arnavkj95/candidate-generation-and-luna16-preprocessing, 2017.

[12] Hara Kensho. 3d-resnets-pytorch. https://github.com/jaxony/unet-pytorch, 2017.

[13] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, Matthew P. Lungren, and Andrew Y. Ng. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. 2017.

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. 2015.

[15] Bram van Ginneken, Arnaud A. A. Setio, and Colin Jacobs. Off-the-shelf convolutional neural network features for pulmonary nodule detection in computed tomography scans. 2015.

[16] He Yang, Hengyong Yu, and Ge Wang. Deep learning for the classification of lung nodules. 2016.