CS230

Detection of Early Stage Lung Cancer from CT-Generated 3D Lung Volumes

Ankit S. Baghel
Department of Computer Science
Stanford University
abaghel@stanford.edu

Lucas Ramos
Department of Computer Science
Stanford University
ramosl@stanford.edu

Abstract

The goal of this project was to to convert computerized tomography (CT) scans into 3-dimensional volumes that deep learning architectures could use to detect early stage lung cancer. A secondary goal is to see if relevant features for cancer detection are maintained through the downsizing of stored volumes.

1 Introduction

Lung cancer is one of the most common types of cancer and the leading cause of cancer death [1]. Early detection of lung cancer enables proactive treatment that vastly improves the chances of survival. However, early stages of cancer are difficult to detect due to the small size of tumors at this stage. Often, lung cancer is not easily detectable until it has progressed significantly [1]. It may be possible to use CT images of the lung to train a deep learning algorithm that can detect lung cancer during its earlier stages. If successful, this algorithm could be used in Computer-aided diagnosis (CAD) that helps improve accuracy of early diagnosis in at risk patients.

Currently, CT scan data comes in the form of cross section slices that deep learning frames works can parse through one at a time to detect tumors that are malignant. However, this approach requires storage and processing of large amounts of data that makes training expensive and slow. We aimed to use 2D and 3D CNNs to explore whether reduced-size 3D volumes generated from the CT slices for a patient are sufficient for binary classification of cancer in patients with and without early stage lung cancer.

2 Related work

Many current machine learning algorithms consider individual CT scan slices when looking for tumors [2,3,4]. Using individual slices results in less computationally expensive training because of smaller data input that the model has to consider compared to the 3D model that we created to consider entire volumes. However, we wanted to determine if looking at multiple slices concurrently in the form of a 3D volume would improve learning.

Another popular technique is the use of Artificial Neural Networks [5,6]. These models are inspired by networks of biological neurons, in which the neurons compute output values from inputs. These models' goal is to solve problems in a similar way as the human mind would. Since tumor detection in lung CT scans is a task that is normally performed by humans, this approach is intuitive. Unlike humans, however, deep neural networks could learn to consider multiple slices at a time and learn

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

unintuitive features that improve the overall accuracy of classification. Because of this, we aimed to explore 3D CNNs.

3 Dataset and Features

Our data comes from the Kaggle Data Science Bowl 2017 which contains lung CT scans of 2100 patients [7]. Each patient was labeled either 0 for no cancer diagnosis within a year or 1 for cancer diagnosis within a year. Approximately 70% of the patients in the dataset did not have early stage cancer whereas the remaining 30% percent did. We attempted to address the class bias within the data by weighting the loss from misclassifying each class differently such that misclassifying a patient with lung cancer resulted in a greater loss than misclassifying a patient without lung cancer. Each patient folder contains a variable number of CT scan slices. Each slice is a 512 x 512 image provided in the DICOM format. After preprocessing the images as required, we ran this data set through our CNNs to classify CT scan collections for early stage cancer with 80% in the training set and 20% in the validation set. We evaluated our results quantitatively by examining binary cross entropy loss and accuracy for both training and validation sets.

4 Methods

Preprocessing

Our preprocessing steps for the DICOM files for each patient were based on a Kaggle kernel from the competition the data set came from [8]. For each patient in the data set, we start by stacking all of the CT slices onto each other to create a 3D image of the lung. Next, we convert the pixel values of the 3D image to Hounsfield Units (HU), the measurement of radiodensity used for CT scans. Since we are not interested in bones, air, and other non-lung tissue objects that have radiodensity, we used thresholding to segment the lung tissue from the rest of the 3D image and reduce overall noise of the data. Lung tissue generally has HU of -500 whereas air has HU of -1000 and bone has HU of $+700$[8]. Therefore, to perform thresholding, we set any pixels with HU value of -320 or higher as 1 and all else as 0. After segmentation, the pixel values of the 3D lung images are normalized and zeroed to be between 0 and 1. Finally, we downsize the 3D images using the Tensorflow resize_images function with an area interpolation filter before padding images as required to construct a NumPy array for both the training and validation sets.

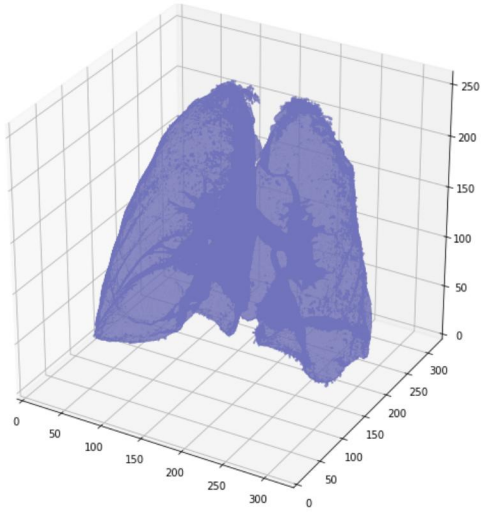

Figure 1: Preprocessed lung

Processing/Architectures

When looking for a framework we had difficulty finding a neural network set up for three-dimensional network, so we instead opted for one that works for two-dimensional networks and work from there. When searching, we settled for a project on github named cnn finetune as it already had support for transfer learning. We knew that was a feature we wanted to try and use so this repository was a good starting point. In addition to this, the repository had a couple of pre-built models to use. The ones that we decided to experiment with using were their densenet and their resnet models, as these were models that we were familiar with because of their use in class. To make our dataset which was in 3D compatible with this model in 2D, we decided to flatten the vectors into a 2D shape instead.

Additionally, we developed two 3D CNN architectures from scratch. The first was based on the UNet architecture described in [9]. First, the 64x64x64 volumes are run through 3D convolutional layers while being downsampled to a size of 8x8x8. Then, downsampling layers are replaced with upsampling layers so that the output of the last 3D convolutional layer is of size 64x64x64. Lastly, the output from this 3D convolution is run through a batch normalization layer, followed by a flattening layer, two fully-connected dense layers, and finally a sigmoid activation layer that makes a prediction.

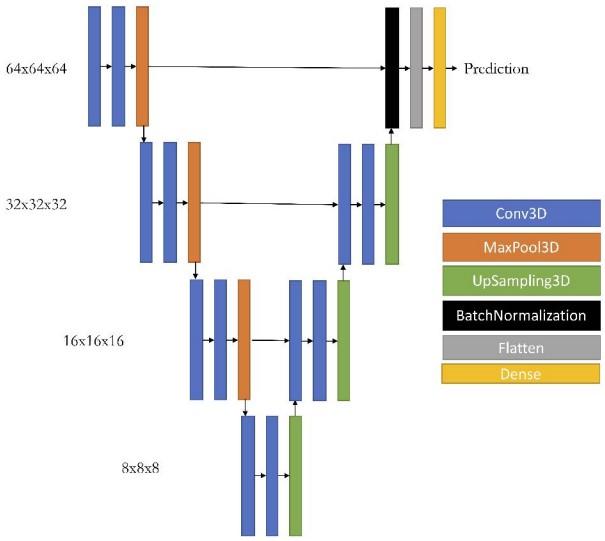

Figure 2: 3D UNet Model

Because UNet implementations are better optimized for segmentation tasks rather than binary classification, we opted to implement a more traditional 3D CNN that was deeper. The second 3D CNN removes the upsampling steps of the UNet implementation and instead uses additional 3D convolutional layers.

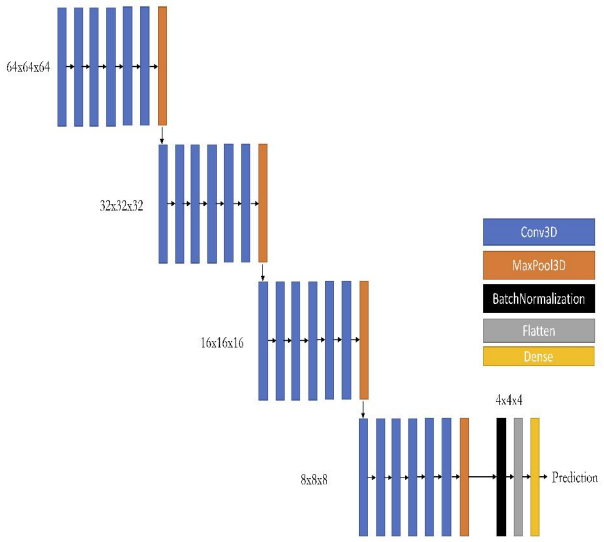

Figure 3: 3D CNN Model

5 Experiments/Results/Discussion

After running our models, the best loss and accuracy achieved for each model is as follows:

	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
ResNet-152	0.5926	0.7295	0.6239	0.7000
DenseNet-161	0.6007	0.7240	0.6206	0.7000
3D-UNet	23.636	0.7373	34.1301	0.6975
3D-CNN	1.4112	0.7373	0.1390	0.6975

Figure 4: Best Training and Validation Loss and Accuracy

We also tuned our hyperparameters to optimize our algorithms:

3D UNET					
Learning Rate	*Drop_rate*	*Training Loss*	*Training Accuracy*	*Validation Loss*	*Validation Accuracy*
0.045	0.3	29.6355	0.7373	34.1301	0.6975
0.045	0.5	35.2647	0.2627	33.3594	0.3025
0.045	0.1	29.6355	0.7373	34.1301	0.6975
0.0003	0.3	35.2647	0.2627	33.3594	0.3025
3D CNN					
Learning Rate	*Drop_rate*	*Training Loss*	*Training Accuracy*	*Validation Loss*	*Validation Accuracy*
0.045	0.4	1.4112	0.7373	0.139	0.6975
0.045	0.5	1.4075	0.7342	0.1392	0.6975
0.045	0.3	1.6793	0.2627	0.139	0.6975
0.045	0.1	1.6443	0.3196	0.139	0.3025

Figure 5: Hyperparameter Tuning

In looking at our results, it seems that all of our models performed roughly equivalently on the dataset ($\tilde{7}0\%$ accuracy). This was unexpected since we tested both 2D and 3D models, yet they yielded the same results. Later analysis of the 3D-CNN revealed that specificity was 1.0 and sensitivity was 0, which means that the classifier was not learning useful features and was instead taking advantage of the class bias of the data set. This was found to be the case across all of our tested models.

It is likely that downsizing all dimensions to 64 resulted in loss of detectable cancer features. To handle this issue, we re-ran the 3D-CNN on the dataset before it was downsized completely to see if any features could be learned. However, this was not sufficient to prevent overfitting to a single class (result not displayed). By altering hyperparameters like learning rate and dropout rate, we could push the classifier to focus entirely on specificity (all 0s) or on sensitivity (all 1s), but we could not get the classifier to converge in between. It is likely that the class bias in the dataset presents two local minima in the loss function in which the classifier gets stuck once it converges to them.

6 Conclusion/Future Work

Since we may have lost detectable features from downsizing the volumes, we should re-run the initial lung volumes through our architectures to determine if any detectable features may be learned. However, the initial size of the volumes (400 x 400 x 500) are too large to be run in their entirety on available machines. Therefore, we should explore how to detect regions of interest so that the subspace the algorithms must search over are much smaller. Segmentation algorithms might present the best method of detecting these regions of interest. Additionally, we could add residual blocks to our 3D-CNN to improve its ability to learn new features over additional layers.

7 Contributions

Ankit Baghel was primarily responsible for downloading the Kaggle dataset, preprocessing the slices into 3D volumes with desired dimensions, and generating the training and validation sets as NumPy arrays. He also implemented the 3D UNet and 3D CNN architectures used in the project using Keras. Lucas Ramos was primarily responsible for performing the fine tuning on the cnn finetune Github code that was used for transfer learning. Both made equal contributions to the final paper and poster.

8 Contributing Code

All code used can be found on Gradescope and at the following Github repository https://github.com/ramosl/lungclassification

References

[1] “Lung Cancer.” American Cancer Society, www.cancer.org/cancer/lung-cancer.html.

[2] Madabhushi, Anant, et al. "Automated detection of prostatic adenocarcinoma from high-resolution ex vivo MRI." IEEE transactions on medical imaging 24.12 (2005): 1611-1625. APA

[3] Madabhushi, Anant, et al. "Comparing ensembles of learners: Detecting prostate cancer from high resolution mri." International Workshop on Computer Vision Approaches to Medical Image Analysis. Springer, Berlin, Heidelberg, 2006.

[4] Nattkemper, Tim W., et al. "Evaluation of radiological features for breast tumour classification in clinical screening with machine learning methods." Artificial Intelligence in Medicine34.2 (2005): 129-139.

[5] Cruz, Joseph A., and David S. Wishart. "Applications of machine learning in cancer prediction and prognosis." Cancer informatics 2 (2006): 117693510600200030.

[6] Kourou, Konstantina, et al. "Machine learning applications in cancer prognosis and prediction." Computational and structural biotechnology journal 13 (2015): 8-17.

[7] Kaggle, “Data Science Bowl 2017.” https://www.kaggle.com/c/data-science-bowl-2017.

[8] “Full Preprocessing Tutorial | Kaggle.” Countries of the World | Kaggle, www.kaggle.com/gzuidhof/full-preprocessing-tutorial.

[9] Ronneberger, Olaf. “Invited Talk: U-Net Convolutional Networks for Biomedical Image Segmentation.” Informatik Aktuell Bildverarbeitung Für Die Medizin 2017, 2017, pp. 3–3., "doi:10.1007/978-3-662-54345-0_3.

[10] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané,

Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[11] Chollet, François. Keras. https://github.com/fchollet/keras, 2015.