Lung Tumor Segmentation via Fully Convolutional Neural Networks

Austin Ray
Stanford University
CS 231N, Winter 2016
aray@cs.stanford.edu

Abstract

Recently, researchers have made great strides in extracting traits of lung tumors from Computerized Tomography (CT) scans in order to more accurately diagnose and better treat patients. Logically, these lung tumor features are extracted only from pixels that lie in and around the tumor. Currently, figuring out which voxels 'belong' to the tumor (a problem which we will refer to as 'tumor segmentation') is done manually by trained radiologists - an unfortunate inefficiency in the process. Algorithms based on traditional computer vision techniques have proven too inaccurate to be useful. Building on recent image segmentation breakthroughs at the Berkeley Vision and Learning Center [1], this paper presents a fully convolutional neural network that – using an initial bounding box around the tumor – can segment a tumor in two dimensions at a mean testing Dice Score [Figure 3] of 83%.

1. Introduction

1.1. CT Scans

A CT scan of a patient's chest is usually one of the first tests a radiologist will do to see if a patient has a lung tumor. For reference, a single CT scan image (slice) is 512x512x1 pixels (grayscale) and there are usually ~200 slices per scan – we say that these 200 slices compose the 'CT volume'. If a lung tumor does appear on the CT scan, radiologists can then analyze the scan further to figure out the type, size, age, and danger of the tumor - in other words, the tumor's features. While trained radiologists can ascertain quite a bit about a tumor with their eyes alone, there exist a wealth of other features of the tumor that can only be gleaned through computational methods.

1.2. Importance of Segmentation

The computational feature extractor is given some collection of 2d regions on various slices in a CT volume. The extractor only looks inside these regions for features. Ideally, these regions will be composed only of tumor pixels and pixels close (i.e. relevant) to those tumor pixels – that is, we don't want to extract features from a rib on the left side of the lung when what we really care about is a tumor on the right side of the lung. This is the reason why accurate tumor segmentations are important – to provide a pure set of pixels for the feature extractor to analyze.

1.3. Inputs and Outputs

We will refer to the neural network discussed in this paper as the Tumor Segmentation Fully Convolutional Network, or TS-FCN.

The input to the TS-FCN is a 100x100x3 image with a tumor somewhere in it. All inputs are assumed to have tumors in them (the focus here is not on detection). To form each 100x100 image, a liberal bounding box is drawn around the tumor on a 512x512 CT slice. Bounding boxes are 100x100 at minimum in order to retain spatial context from the lung. Next, the area inside this bounding box is cut out from the 512x512 image and resized down to 100x100 if necessary. For example, if a tumor is actually ~80x80, we might make a 160x160 liberal bounding box, cut out this box from the 512x512 source CT image, and then downsize to 100x100 to get an input to the TS-FCN.

Since the original 512x512 slices have only 1 color channel, we copy this channel to 2 additional color channels for our 100x100 bounding box images in order to utilize pre-trained convolution layers that only accept 3-channel images.

The output of the TS-FCN is a 100x100x1 image. A pixel in the image is 1 if the TS-FCN predicted that pixel to be a tumor, and 0 otherwise.

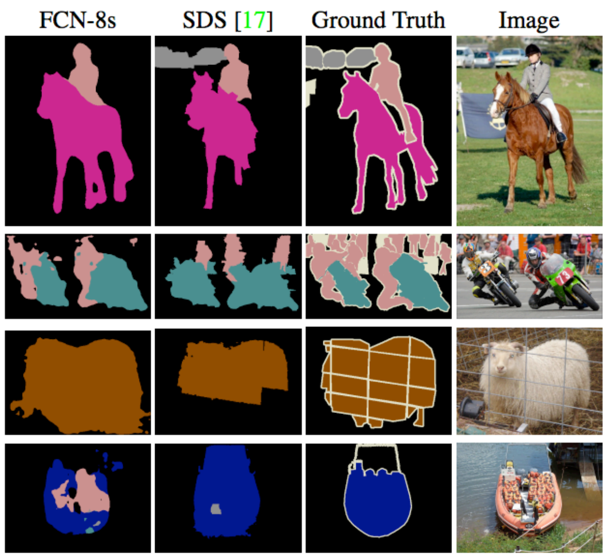


Figure 1: Output of previous state-of-the-art segmentation system (SDS) and of deep jet compared to the ground truth segmentation.

2. Related Work

2.1. Manual Segmentation

Since radiology is a field where errors are extremely costly, it is vitally important for radiologists to use the most accurate tools available to them, no matter the cost or time. Consequently, since the human eye is currently the best segmentation algorithm out there, all lung tumor segmentation for the sake of feature extraction is done by hand. This can take as much as 10 minutes per patient, however, which is quite a lot of time for radiologists.

Another issue with manual segmentation is that there is often no consensus about which regions in an image contain features worth extracting. Some radiologists are conservative with their segmentations and perfectly outline the solid portion of a tumor. Other radiologists are more liberal and tend to include surrounding regions, the idea being that there is important information contained in those regions as well. In fact, recent work by a graduate student in my lab shows that segmentations can differ by as much as 20% between radiologists (paper release pending), which can be disastrous for algorithms down the pipeline that rely on accurate segmentations.

The reasons why it is important to develop computational segmentation algorithms are therefore (1) to save radiologists' time, and (2) to provide precise segmentations.

2.2. Windowed Segmentation

One way that researchers have approached segmentation is by using windowed neural network architectures. A group out of IDSIA recently implemented this architecture in order to perform two-class membrane segmentation on 2D slices from brain scans [2]. The neural network in that paper takes in a square image centered on a pixel and outputs a class for the center pixel. You can attain a full segmentation for an image by feeding in a series of windows centered on all pixels in the image. In other words, you would have to feed all 10,000 windows from a 100x100 image into the net in order to get a full segmentation for that image.

One interesting issue this paper deals with is heavy class imbalance in a 2-class segmentation problem. Heavy class imbalance is a problem because the neural network can simply guess that a pixel belongs to the much-more-frequent class every time and automatically get extremely low loss – a local minima that is hard to escape from. For my project, I have to face the same problem, since tumor pixels only compose ~4% of pixels in a 100x100 bounding-box input image. The IDSIA paper cleverly deals with class imbalance by down-sampling pixels from the more frequent class so that the neural network trains on an equal number of pixels from each class. After training, the IDSIA group places a calibration layer on top of their net's output for scaling probabilities to match the statistical reality of the two classes in the data.

Unfortunately, since I ended up going the fully convolutional route, it was not possible for me to down-sample inputs, since every pixel in an image was necessary to classify the other pixels in that image. However, the IDSIA group inspired me to implement a weighted loss function to counter class imbalance, the implementation of which I will talk about later.

Although this paper reports extremely high accuracy and its methodology seems straightforward, its reported time of computation per pixel is over ten times that of the paper I will mention in section 2.3. Due to time constraints for this project, I chose not to pursue this methodology.

2.3. FCNs for Semantic Segmentation

Most of the inspiration for this project comes from a recent paper out of the Berkeley Vision and Learning Center [1]. The authors of this paper outline a neural network architecture, which they call 'deep jet', that takes in a 2D image from the PASCAL VOC dataset [6] as input and outputs a 21-class semantic segmentation of that image (see examples in Figure 2).

The architecture differs from past segmentation architectures in that it is 'fully convolutional', meaning it has no inner product (linear) layers – only convolutional layers. This allows the network to take in 2D inputs of arbitrary size and return an output of the same size, where each pixel is a prediction for its corresponding pixel in the input image.

Deep jet's ability to process all pixels in an image at once make it 1-2 orders of magnitude faster than windowed segmentation. Additionally, deep jet enjoys larger receptive fields than the windowed segmentation architecture, as it is not limited by the chosen window size. This added context can be quite useful in segmentation.

3. Methods

All neural networks used in this project were built and executed using the Caffe framework [5].

3.1. Net Architecture

The TS-FCN used in this paper is much like the 'deep jet' architecture used in the BVLC paper [1], but slightly less complicated and smaller. The net architecture is outlined below:

(1) Conv64 – ReLU – Conv64 – ReLU – MaxPool
(2) Conv128 – ReLU – Conv128 – ReLU – MaxPool
(3) Conv256 – ReLU – Conv256 – ReLU – MaxPool
(4) Conv4096 – ReLU – Dropout0.5
(5) Conv4096 – ReLU – Dropout0.5
(6) Conv2
(7) Deconv8x – Crop
(8) Softmax – WeightedMultiClassLoss

All Conv64, Conv128, and Conv256 convolution layers use size 3 filters with a stride of 1. All use zero padding of 1 except for the first Conv64, which uses zero padding of 32 so that the image isn't too small after the 3 max-pools. The first Conv4096 uses a filter size of 7. The second Conv4096 and Conv2 both use filter size 1 – these are the layers that make the net 'fully convolutional', as they mimic fully connected/linear layers applied to each pixel. Conv2 gives a score for each of the 2 classes for each pixel.

At this point, the image has been max-pooled 3 times and is thus about 8x smaller than it started – 2x15x15, to be exact. The deconvolution layer uses a filter size of 16 and a stride of 8, meaning the result is 2x128x128 (Solve [(H – 16)/8 + 1 = 15] for H). This result is then cropped in the center to give a 2x100x100 image.

The softmax function is then applied to each pixel's 2 class scores to give class probabilities. These probabilities are then fed into a weighted multi-class loss layer where each pixel is treated as a sample in a mini batch (100x100 image is a 10,000 member mini batch), with the total loss equaling the average loss over every pixel in the image. To account for the fact that tumor pixels are only 4% of all image pixels in my dataset, I weighted the contribution of each ground-truth tumor pixel by 0.96, and the contribution of each ground-truth non-tumor pixel by 0.04. A regularization weight of 5e-4 was used in the loss equation in order to increase performance on the validation set. The final loss equation I used can be seen in Figure 2.

$$Loss(image) = \left[0.96 \sum_{\{i \in image | y_i = tumor\}} L_i \right] + \left[0.04 \sum_{\{j \in image | y_j \neq tumor\}} L_j \right] + reg_{image}$$

$$L_i(image) = -\log \left(\frac{e^{f_{y_i}}}{\sum_{j \in \{0,1\}} e^{f_j}} \right)$$

Figure 2: Weighted softmax loss function. y_i is the ground truth value for pixel i - either 0 (not tumor) or 1 (tumor). f_j is the score for class j for the pixel (output from layer (7)), where j is 0 or 1. reg_image is a weight regularization term.

3.2. Learning

Weights for layers (1), (2), and (3) were transferred from the corresponding layers in source [1], which were available at [3]. These weights can be interpreted as producing a feature set for each pixel from any given image, with the rest of the net learning to use these features to do segmentation. These pre-trained layers were trained at 1/10th the learning rate of the other layers. The deconvolution layer was initialized with bilinear interpolation weights and its learning rate was set to 0 (it therefore just served as a basic image up-sizer).

ADAM was used as an update rule, with learning rate 1e-6. The learning rate was multiplied by 0.9 four times per epoch. The net was trained for around 5 epochs. A batch size of 1 image was used.

3.3. Performance Metrics

The main performance metric used was Dice Score, which is like intersection over union. To calculate the dice score for a given prediction/ground-truth pair, you count up all the pixels they both predicted were tumors, divide that by the combined sum of tumor pixels predicted by both, and multiply by 2. This gives a value between 0 (no agreement) and 1 (full agreement). The equation can be seen in Figure 3 below. This metric is a much better evaluator of performance than pure accuracy, as pure accuracy will be high for all images due to the prevalence of non-tumor pixels.

$$QS = \frac{2C}{A+B} = \frac{2|A \cap B|}{|A| + |B|}$$

Figure 3: Dice score equation, where A is the set of predicted tumor pixel locations and B is the set of actual tumor pixel locations.

One important thing to note is that 2D dice scores aren't that interesting to us here. What we're really interested in is the dice score over a patient's entire tumor, which spans multiple slices. Thus, all validation dice scores are 3D dice scores, where the numerators and denominators are each summed individually before being divided by each other.

4. Dataset and Features

I have access to CT scans and corresponding radiologist segmentations for 107 patients through the Stanford Radiological Image and Information Processing Lab, where I work [4]. Each patient's scan has around 200 slices in it. Most of these slices have no tumor in them. Each slice is a 512x512 grayscale image. Not all scans were done by the same machine or technician a lot can vary from scan to scan. This is part of the challenge.

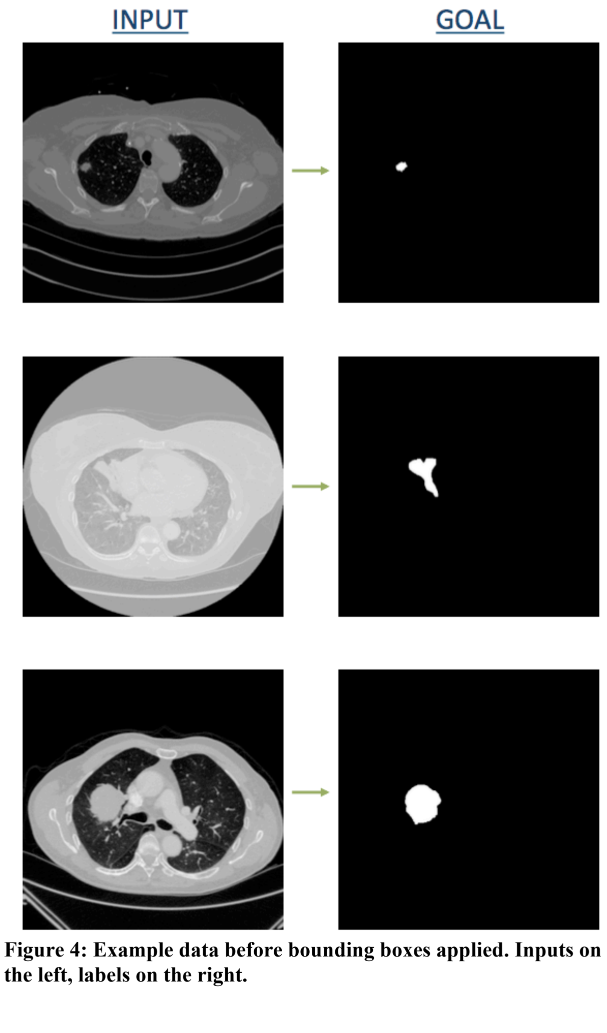


Figure 4: Example data before bounding boxes applied. Inputs on the left, labels on the right.

4.1. Preparing the Data

First, I got rid of any slices that had no tumor in them (as seen when the segmentation for that layer had no 1's (tumor pixels) in it). Next, I created square bounding boxes for each tumor that were about twice as big as the tumor's maximum width and height at any layer in the patient. Any bounding boxes under 100x100 were automatically set to 100x100 and centered on the tumor. Any bounding boxes over 100x100 were fine, but were resized to 100x100 after being cut out in order to standardize inputs. Labels (segmentations) were cut out and resized exactly like their corresponding slices.

The data's grayscale values ranged from -2000 to 4095. I rescaled these values to 0 to 255 in order to be in line with what the first three pre-trained conv layers expect. The images' color channels were then arbitrarily increased to 3 channels instead of 1, copying that 1 channel equally into 3 in order to make the images compatible with the first 3 pre-trained conv layers. Interestingly, using the color channel means from [3] – which has a different mean for each channel – worked better than using the actual mean for the data, which would also be the same across all 3 channels.

The data was then further augmented through mirroring and rotation to produce 8x as much data, giving around 26,000 total images instead of the original ~3200 images. This turned out to help tremendously increasing the validation accuracy of the net.

The data was split into 5 folds – 4 folds with 21 patients and 1 folds with 23 patients. The data was split this way because the evaluation metric is not on a per-image basis but on a per-patient basis, so it is important to have an equal number of patients in each fold. Unfortunately, due to time constraints, cross-validation testing could only be carried out once. However, I intend to execute complete 5-fold cross-validation when I continue with this project.

5. Results and Discussion

5.1. The Good

After training for ~7 hours (~5 epochs), a 3D validation dice score of 0.86 was achieved. Figure 5 shows validation accuracy as a function of 1/10th epochs (e.g. 20 corresponds to score after 2 epochs, 50 to score after 5 epochs – apologies for the plot messiness). The overall accuracy on a per-image basis was ~99% (high, due to 96% of pixels being background).

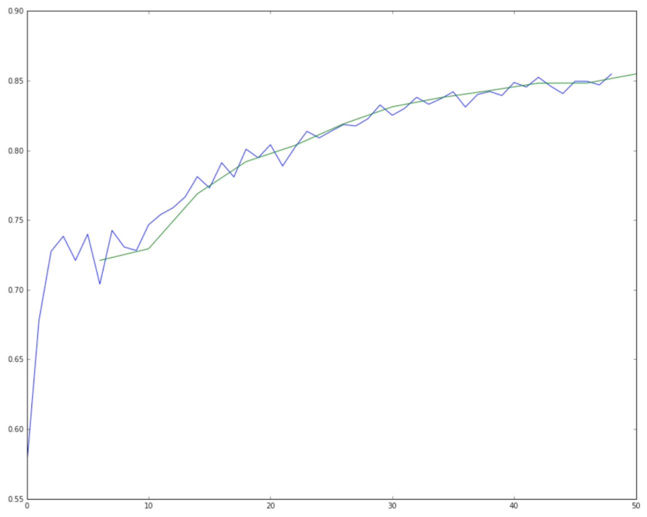


Figure 5: Average validation 3D dice score over all validation patients as a function of number of iterations (10 = 20000 iterations, 20 = 40000 iterations, etc.)

Some example outputs of the resulting model on the validation set can be seen in Figure 6. Although we can see a few mistakes here, the model seems to be highly accurate on the whole. It should be noted that these images are some of the higher-end 2D segmentations, with 2D dice scores in the low to mid 90s.

Interestingly, we can see that the predicted segmentation is often more blob-y than the ground truth segmentation. My hypothesis is that this is a result of the high regularization used in order to prevent against overfitting in the training set.

Another interesting point is that, while larger mini-batch sizes are usually better, they consistently performed worse for this model, with a mini batch size of 1 giving by far the best performance. I posit that this is because there is so much to learn from each image (a 10,000-pixel mini-batch itself) that grouping images together in a batch makes the net try to be too broad in its improvements. With just one image per batch, the net can make smaller adjustments to the net in order to compensate for specific cases, ultimately leading to a better result.

With a validation dice score of 0.86, these segmentations become well within the margin of the 20% variance between radiologists. As such, it is my opinion that this model is definitely worth pursuing further in order to develop a truly useful tool for radiologists going forward.

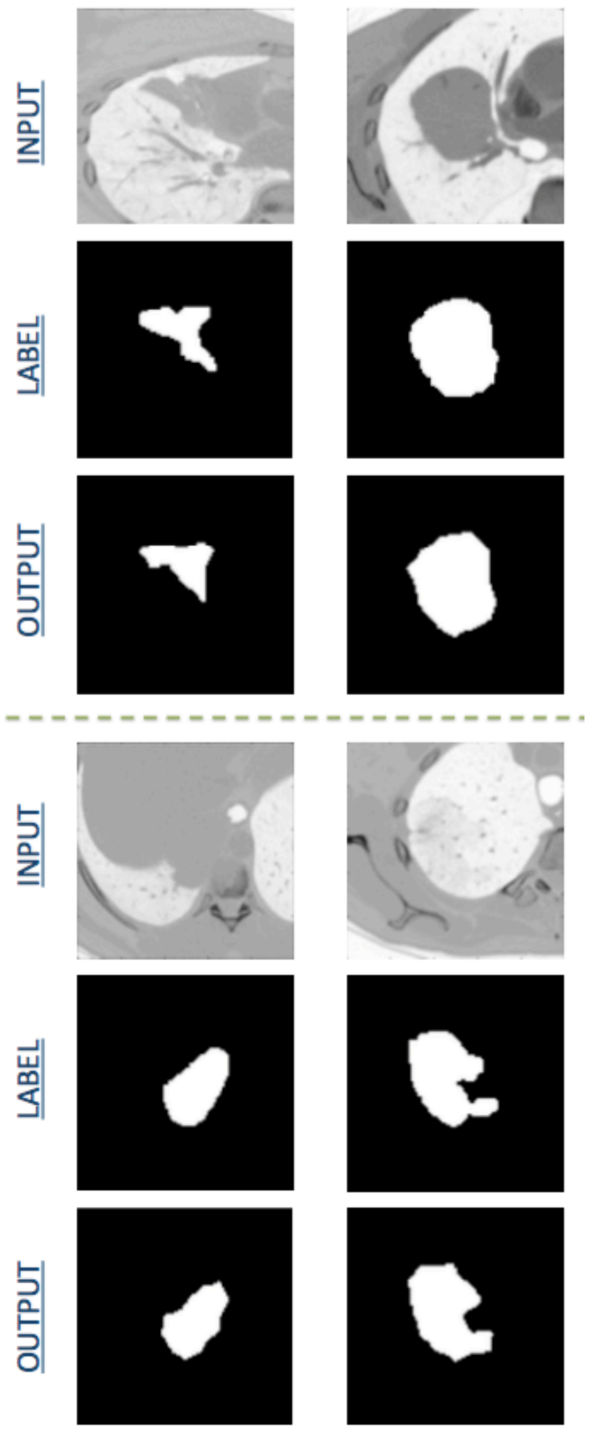


Figure 6: Model evaluated on images from the validation set after 5 epochs of training.

Another interesting thing to look at is what an image looks like after the score, deconv+crop, and softmax layers (Figure 7). As said previously, the resolution of the image before deconvolution is only 15x15. The values for the 'yes tumor' score for each pixel are displayed in figure 7. We can see that although the 15x15 image looks choppy, it actually carries enough information to be upscaled and converted to a very nice pre-threshold segmentation (the original image, true segmentation, and predicted segmentation after thresholding can be seen in the top left series of Figure 6).

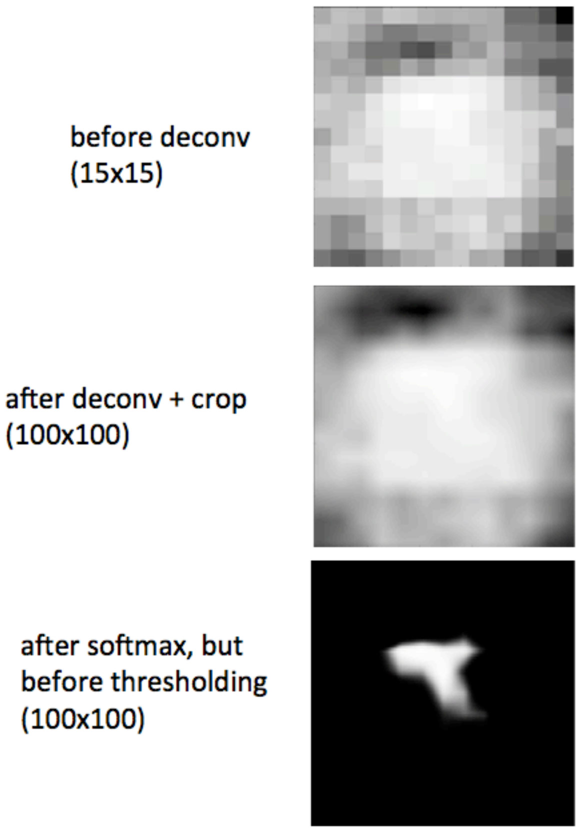
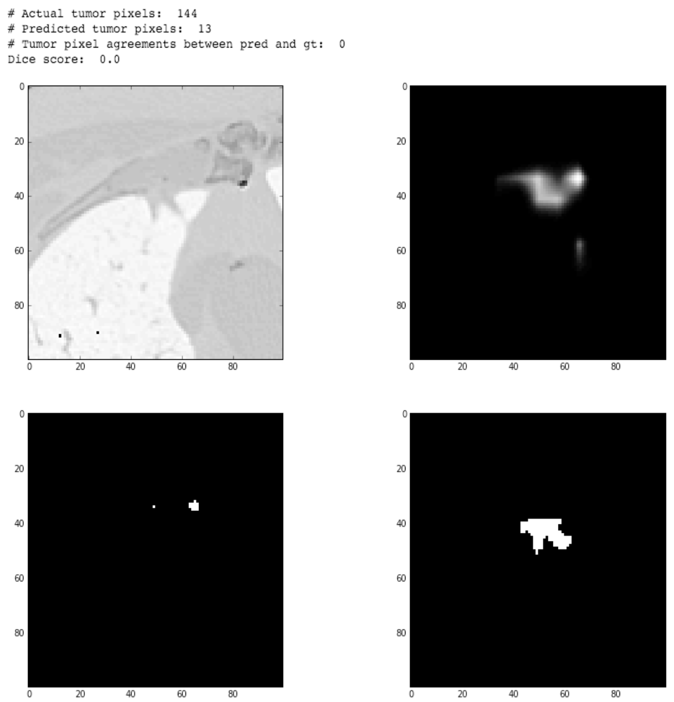

before deconv (15x15)
after deconv + crop (100x100)
after softmax, but before thresholding (100x100)

Figure 7: Data as it passes through last layers and gets transformed into a segmentation.

5.2. The Bad

There are many areas where this framework fails. Besides the blob-iness mentioned previously, the network seems to lack confidence for some tumors (Figure 8). By this, I mean that the network sees the tumor and gives its pixels some positive probability of being tumor, but isn't convinced and ends up leaving them out of the final segmentation. This might be corrected by increasing the weight on the weighted loss in order to more harshly penalize for missing ground truth tumor pixels.



```
# Actual tumor pixels:  144
# Predicted tumor pixels:  13
# Tumor pixel agreements between pred and gt:  0
Dice score:  0.0
```

Figure 8: (top left) Original image, (top right) Image after softmax layer, (bottom left) Predicted segmentation after thresholding, (bottom right) Ground truth segmentation

The model also tends to sometimes think bones, tissues, and organs are tumors (Figure 9). This is a fair mistake, and one that I would make, but at the same time I would expect the model to recognize that that area's textures are not tumor-like and ignore that area. I think this is a symptom of a bigger problem: lack of data. With only 107 patients, I only have access to 107 tumors, and it's likely that some of these tumors are unique, given the genetic variety seen in tumors. Thus, while we might expect the training set to be 'representative' of the validation set, this might not be the case, and some tumors in the validation set could be unlike anything seen in the training set. With more patients, this might not be as much of a problem.

One other possible solution besides 'more data' (which I'm pretty sure is everyone's solution to every machine learning problem), is to allow different images belonging to the same patient to be in both the training and validation sets. Of course, we would need to change the performance metric from 3D dice score to 2D dice score, but we might see an overall increase in performance due to making the training set more "representative".

```
# Actual tumor pixels:  581
# Predicted tumor pixels:  885
# Tumor pixel agreements between pred and gt:  477
Dice score:  0.650750341064
```

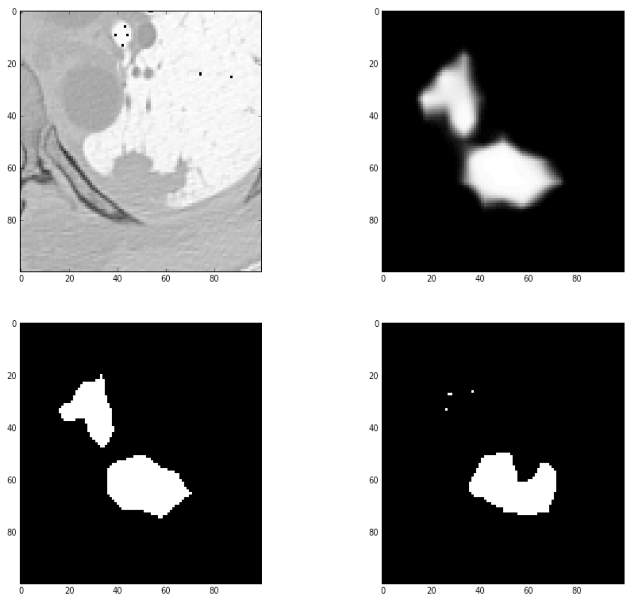


Figure 9: (top left) Original image, (top right) Image after softmax layer, (bottom left) Predicted segmentation after thresholding, (bottom right) Ground truth segmentation.

6. Conclusions and Future Work

Based on the results seen here, we can see that fully convolutional networks are fast, powerful systems for predicting segmentations. I plan to continue my work with FCNs for lung tumor segmentation next quarter for my lab, exploring the possibility of 3D convolutions across CT slices, attempting to segment a full 512x512 image instead of a window around the tumor, and finishing implementing the entirety of the Deep Jet architecture, including the combination of predictions from various resolutions in the net.

7. References

[1] http://arxiv.org/abs/1411.4038; Fully Convolutional Networks for Semantic Segmentation; By Jonathan Long, Evan Shelhamer, and Trevor Darrell; BVLC, 2014

[2] http://people.idsia.ch/~juergen/nips2012.pdf; Deep Neural Networks Segment Neural Membranes in Electron Microscopy Images; By Dan C. Ciresan, Luca M. Gambardella, Alessandro Giusti, and Jurgen Schmidhuber; IDSIA, 2012

[3] https://gist.github.com/shelhamer/80667189b218ad570e82#file-readme-md

[4] http://med.stanford.edu/riipl.html

[5] https://github.com/BVLC/caffe

[6] http://host.robots.ox.ac.uk/pascal/VOC/