3D Convolutional Neural Network for Brain Tumor Segmentation

Bora Erden
Stanford University
650 Serra Mall,
Stanford, CA, 94305
berden@stanford.edu

Noah Gamboa
Stanford University
650 Serra Mall,
Stanford, CA, 94305
ngamboa@stanford.edu

Sam Wood
Stanford University
650 Serra Mall,
Stanford, CA, 94305
swood95@stanford.edu

Abstract

We designed a three-dimensional fully convolutional neural network for brain tumor segmentation. During training, we optimized our network against a loss function based on the Dice score coefficient, which we also used to evaluate the quality of the predictions produced by our model. In order to accomodate the massive memory requirements of three-dimensional convolutions, we cropped the images we fed into our network, and we used a U-NET architecture that allowed us to achieve good results even with a relatively narrow and shallow neural network. Finally, we used post-processing in order to smooth out the segmentations produced by our model. Ultimately, our best model achieved a Dice score of 0.71 on our test set, making our model nearly as effective as state-of-the-art architectures.

1. Introduction

We set out to build a convolutional neural network to classify tumors and tumor subsections in MRI brain images. Medical image analysis is a very important field, and we believe that computer algorithms have the potential to reproduce or even improve upon the accuracy of human experts. Using algorithms to automate medical image analysis could save time and money for hospitals and patients, and improved accuracy would be a great benefit to cancer patients.

Given an input of an MRI brain volume, our neural network outputs a semantic segmentation of the volume that separates the tumor from the rest of the brain. The final output is the same shape as the input, but each pixel of the output, rather than containing visual information, contains the unscaled probability that the corresponding pixel in the input belongs to the tumor. We generate our predicted segmentation by binarizing the predicted probabilities generated by our model, labeling each pixel with predicted probability greater than 0.5 as part of the tumor, and each pixel with probability below 0.5 as non-tumorous. We then evaluate the segmentation produced by our network using the DICE coefficient between the predicted segmentation and the ground-truth label. The Dice coefficient between two sets X and Y is calculated as:

$$Dice(X, Y) = \frac{2|X \cap Y|}{|X| + |Y|}$$

2. Background

Brain tumor segmentation using convolutional neural networks has outstripped the results of generative models – which rely on hand-coded features based on medical knowledge about the appearance of healthy and unhealthy tissue in different parts of the brain – and discriminative models such as SVMs and decision forests [1–3]. Moreover, fully convolutional neural networks, which substitute any final fully-connected layers for convolutional layers, have been shown to achieve the same results more efficiently than architectures using fully-connected layers [1, 2, 4]. These results inspired our decision to approach our problem with a fully-connected convolutional neural network.

Extensive previous work has been done using fully-connected CNNs to segment brain tumors and other medical data. From our review of the literature, we have concluded that 3D CNNs produce significantly better results than 2D CNNs on segmentation problems that, like ours, are three-dimensional [5–7]. However, 3D CNNs are extremely computationally expensive and notoriously difficult to train. Some authors have addressed this difficulty by training on patches extracted from 3D volumes, rather than training on the full volumes in their training set [8, 9]. Other authors have suggested using architectures that, much like ResNet [10], feed forward the outputs of earlier layers to later layers, making it easier to back-propagate gradients and allowing for models to achieve higher accuracy while demanding less memory [6, 11, 12]. Inspired by these results, we

decided to implement a three-dimensional CNN with a U-Net architecture, where the outputs of early down-sampling convolutional layers are concatenated to the outputs of later up-sampling convolutional layers.

Our review of the literature also convinced us that post-processing would improve our results. While a well-trained fully convolutional network can produce segmentations that are very close to ground-truth, these segmentations often include extraneous pixels that are either labeled as part of the tumor even though they are far away from other such pixels or labeled as not part of the tumor even though they are clearly inside of the segmented tumor. Post-processing can help to 'smooth' output segmentations and reduce error from these extraneous mislabeled pixels. Some papers used conditional random field post-processing, which uses pairwise relationships between pixels to maximize label agreement between similar pixels [5, 7, 13]. Other papers have used simpler methods, such as separating contiguous regions of pixels with the same label into "blobs," and changing the labels of any blob with a size below a certain threshold [14]. Some papers even used a simple convolutional filter [15].

Finally, our review of the secondary literature, and especially of papers that have tackled the BraTS Multimodal Brain Tumor Segmentation Challenge in past years, allowed us to establish a benchmark for the success of our model. The best-performing models achieve a Dice score of 0.85-0.9 for tumor segmentations on our dataset [1, 5, 16]

3. Dataset

Our dataset consists of 285 brain volumes, each consisting of 155 two-dimensional slices. These volumes are clinically-acquired multimodal scans, so that we have access to four different versions of each slice image, where different subsections of the tumor may be more visible in different scans. The labels associated with these volumes are manual annotations approved by neuroradiologists. The dataset was provided by 2017 BraTS Multimodal Brain Tumor Segmentation Challenge `http://braintumorsegmentation.org/` [16]. An example brain slice and associated label is in Figure 1.

For our models we trained on 240 samples with a 20% cross fold validation split. we tested on the remaining 45 slices. We never changed the training and test set because we felt that this would help us preserve the integrity of our model no matter what. For our 3D convolutional model, we did do preprocessing of our data. see subsection 4.1.

4. Methods

We use a fully convolutional neural network. After running the input brain volume through a series of down-sampling max-pooling convolutional layers, we feed the

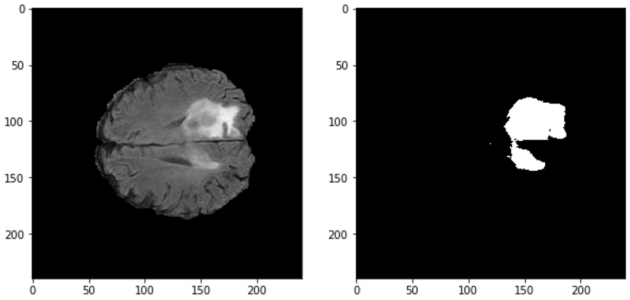

Figure 1. Brain MRI with labeled tumor

data through a series of up-sampling transpose convolutional layers (See Figure 2). The final output of the network is the same shape as the input, but each pixel of the output, rather than containing visual information, contains the predicted probability that the corresponding pixel in the input belongs to the tumor. We generate our predicted segmentations by binarizing the predicted probabilities. Pixels that round up to 1 are labeled as tumorous, and pixels that round down to 0 are labeled as non-tumorous.

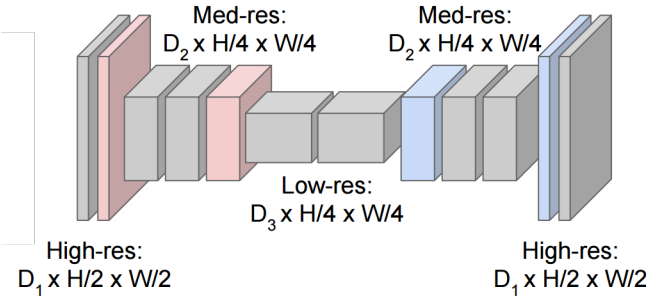

Figure 2. While this does not represent the exact architecture of our model, it demonstrates the principles of a down-sampling-up-sampling convolutional network. (Image taken from CS231n course slides)

After examining several possible loss functions, we decided to optimize our network against a loss based on the Dice score coefficient. 5.1.

Interestingly, the inputs to our network are two-dimensional slices that make up connected three-dimensional volumes, rather than independent two-dimensional images. We therefore tried two different approaches. In the first approach, we fed in single two-dimensional slices into a two-dimensional convolutional neural network and outputted a single slice, with each pixel's value holding the unscaled probability that the corresponding pixel in the input belongs to the tumor. We explored different 2D CNN architectures and hyperparameters in subsections 5.2 and 5.3.

Our second approach used a 3D Fully Convolutional model and trained on entire brain volumes, rather than on individual 2D slices. We assumed that a fully convolutional network using three-dimensional rather than two-dimensional filters should be able to achieve better results. However, the memory demands of three-dimensional convolutions made us consider whether the sacrificed width and depth of our network were worth it.

4.1. 3D Convolutional Model

In order to achieve high accuracy with the constraints of a three-dimensional model, we were inspired to use the technique in [12]. The U-NET model described in that paper, in which the outputs of early down-sampling convolutional layers are concatenated to the outputs of later up-sampling layers, enabled us to have a good starting point for our model (see Figure 3). From this starting point, we manipulated 10 things in order to improve on the U-net Model: Model Layers (activations, filters, number of layers, type of upsampling, kernel size), Data bounds, Loss function, Learning Rate, Dropout Rate, and Optimizer.

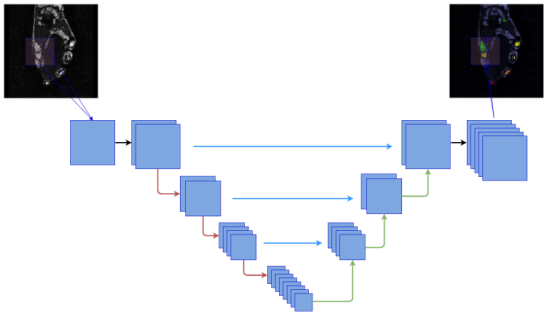

Figure 3. Like the previous figure, this figure does not represent the exact architecture of our model. However, it does illustrate the principles of a U-Net model for a multi-class segmentation problem, in which the outputs of early down-sampling layers are concatenated to those of later up-sampling layers. (Image taken from [11].

We ultimately trained our 3D convolutional models for over 50 hours with different iterations of this architecture. Our 3D convolutional model took in FLAIR from DICOMs and output the segmentation of the entire tumor. This means we limited our dataset from the beginning to only use a single reading of the CT scan instead of the four used to establish the different parts of the tumor. We chose to do this because of the expensive training time of a 3D conv model with many layers and because we wanted to fully test many different architectures. Although we used many different loss functions, our true metric of how well our model performed was determined by the dice score calculated from the predicted probabilities which were the output of our model.

The first thing we did was groom the data so that we could limit the shape of our data for 3D convolutions in order to speed up training and testing. We tried a couple of different pre-processing approaches. The first method we tried was a bounding box over the data set. This turned out to create a lot more frustration with model syntax. The shapes of each scan became $183 \times 179 \times 155$, which was not that great to work with when dealing with upsampling and downsampling sizes. We then resorted to a pseudo-bounding box approach that meant we trimmed the data such that numbers were easy to work with but still removed a significant portion of unnecessary zeros in the dataset (See Figure 4). The shape of the inputs for all our models became $192 \times 192 \times 154$. This was easy to work with in terms of changing kernel sizes and upsampling and downsampling rates.

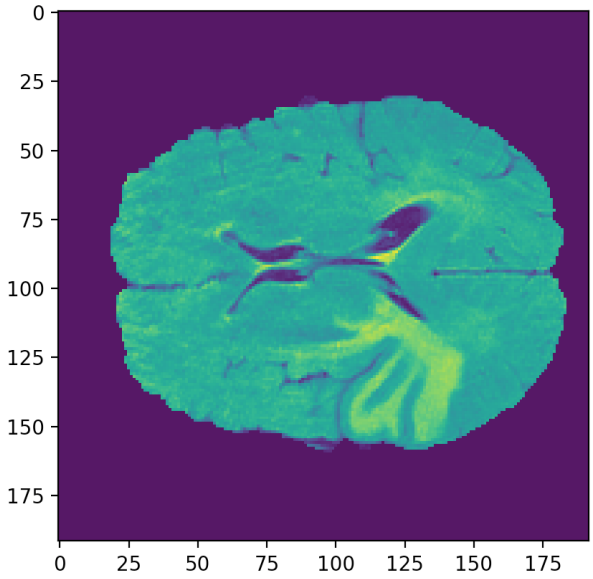

Figure 4. Brain MRI with pseudo crop

In terms of the different models we trained and evaluated, we started with a simple U-net model with only a single convolution per filter size, no concatenations, and only went up to 256 filters and used upsampling instead of transpose 3D convolutions with no dropout and relu activations. We wanted to use this simple and fast trainable model to pick a loss function that would preform well on bigger models. We believed that we could make this extrapolation because of the nature of our training. Given the constraints on our GPU, we could only use a batch size of 1 for larger models. We knew this would affect the way we were moving across the loss landscape and wanted to see how well mod-

els converged for different losses, optimizers, and learning rates for the 3D convolutional model.

The best loss function was a Dice Score based on probabilities rather than values. We tested thoroughly softmax cross entropy loss, cosine loss, dice loss and our new dice probability loss. Additionally, we tried SGD with momentum vs ADAM for dice loss and we found ADAM tended to converge faster (around 5 epochs of training for adam on our small model).

5. Experiments

5.1. Loss function evaluation

We experimented with three different loss functions. First, we tried optimizing our neural network against the softmax cross-entropy objective for our two-class classification problem. Second, since there are far more pixels in our dataset that do not belong to tumors than there are pixels that do, we tried a loss function that attempts to compensate for the imbalanced class distribution in the dataset. We used a sensitivity-specificity loss, which calculates the overlap between predicted and ground-truth tumor pixels (sensitivity) separately from the overlap between predicted and ground-truth non-tumor pixels (specificity). We weighted sensitivity significantly higher than specificity (at a ratio of around 9:1), before attempting to minimize the negative sum of the two. Finally, we tried optimizing against a loss function based directly on the Dice score coefficient. Since the Dice score itself is non-differentiable (since the predicted segmentation results from a non-smoothly-differentiable binarization of the neural network outputs), our objective function was not, strictly speaking, the Dice score for our model's predictions. Rather than using the predicted segmentation, we used the non-binarized predicted probabilities for each pixel. Then, where X is the feature map of predicted probabilities and Y is the ground-truth segmentation, we calculated our loss function as:

$$DiceLoss(X, Y) = -\frac{2 * sum(X * Y)}{sum(X) + |Y|}$$

As can be seen in Figure 5, which shows our results after training a small 2D net for five epochs on our three different loss functions, optimizing against the Dice score loss led to the best results.

5.2. 2D CNN architectures

We experimented with various different depths and widths for our 2D CNN architecture. As can be seen in Figure 6, we found that increasing the number of parameters in our 2D architecture slightly improved our results on the validation set. However, no matter how large we made our 2D net, it was not able to compete with our 3D architectures, even when those architectures used far fewer parameters.

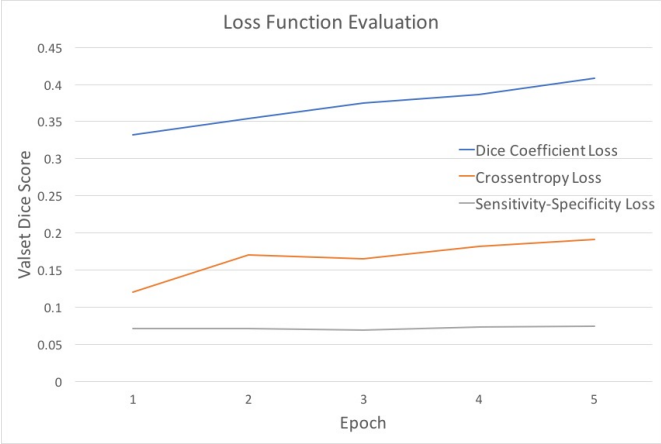

Figure 5. As expected, using the Dice coefficient as the objective yielded the best results.

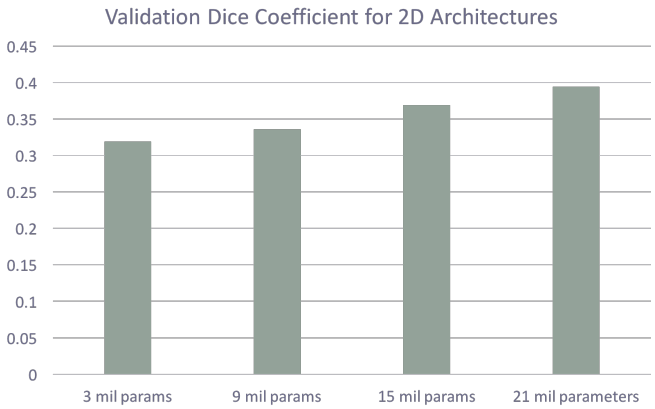

Figure 6. Increasing depth and width of our 2D net improved results, but not enough to compete with out 3D net. Each 2D net was trained for four epochs.

5.3. 2D CNN Hyperparameters

It was difficult for us to thoroughly explore the hyperparameter space for our 3D architecture, given the computational expense of three-dimensional convolutions. However, we did explore many different hyperparameter combinations for our 2D architecture. As can be seen in Figure 7, which displays some of the hyperparaemeter combinations that we tried, we found that a learning rate of 1e-4 and a dropout probability of 0.5 led to the best results. These results gave us some guidance as we performed a limited hyperparameter search for our 3D architecture.

5.4. 3D CNN Architectures

All of our models had a similar structure to U-net. Our first experimentation was with kernel size. In the beginning, all of our kernel sizes for our convolutional layers were $5 \times 5 \times 5$. We found that our output segmentations became

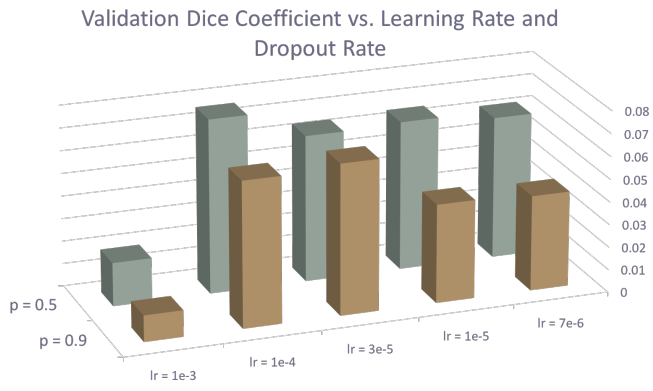

Figure 7. A dropout rate of 0.5 and high learning rate of 1e-4 led to the best validation set dice coefficient after two epochs of training a 2D architecture.

inaccurate around edges and small points. This made sense; we were grouping large sections together so that when upsampling occured, they didn't really map to precise points. After trying a combination of $5 \times 5 \times 5$ and $3 \times 3 \times 3$, we settled on models that used $3 \times 3 \times 3$ exclusively.

The next form of experimentation we did was concatenation and non-concatenation (see 5.5).

The next form of experimentation involved adding more convolutional layers in between up and downsampling. We found that adding more than 2 really did not make a significant difference in the quality of our model but greatly increased our training time. As time was short, we mostly experimented with models with 1 and 2 convolutional layers between up and down sampling.

We also experimented with use of Leaky ReLu vs ReLu for our layers as we thought a problem could be that many of relu units are "dying" during training and creating big changes in dice scores. After running dual experiments with Leaky ReLu and Relu, we decided to keep ReLu as leaky ReLu did not provide much increase in value.

Getting 3D transpose convolutions to work was a pain, however once we implemented them we found they gave a significant boost in Dice score (around .10). For all of our later tests, we used 3D transpose convolutions for upsampling instead of normal upsampling (using extrapolation). We believe this is because pure upsampling is not trainable for our model, so it was simply blowing up our images.

The next thing we experimented with was the number of filters we used. We first started using $[16, 32, 64, 128, 256]$ respectively for each upsampling and downsampling layer. We then experimented with upgrading the number of filters to $[32, 64, 128, 256, 512]$. This more than doubled the amount of training time for our model. However, it again provided with a modest boost in improvement to 3D model (around $.005$). We believe this is because the model was able to learn more high level features than before and able to draw more conclusions about the data.

Finally, our validation performance was never doing as well as our training performance. We decided to use dropout to manage this problem. We experimented with 3 rates of dropout and compared the results: $0, .2, .5$. 0 was simply our baseline model at the time. $.5$ significantly dropped the performance of our model and increased the number of epochs necessary to get to the score that we wanted. This was clearly too large for our model. $.2$ gave a slight improvement to our model and we decided to use this. However, it took 1.5x longer to train our model to get to the same training threshold as before (25 epochs)

5.5. U-Net Architecture

We were inspired by [11] to symmetrically concatenate the outputs of our early downsampling convolutional layers to the outputs of our later upsampling layers. The so-called U-Net architecture was first proposed by [12]. The authors of [12] suggested that combining the high-resolution outputs of early layers with the less informationally dense upsampled outputs of later layers would allow a fully convolutional network to achieve more precise localization. However, our findings did not necessarily support their hypothesis. We trained two identical 3D nets, one with concatenation of early outputs to later outputs, and one without this concatenation. We found that the model with concatenation only slightly outperformed the model without concatenation (See Figure 8). It seems that the primary importance of the U-net architecture is the combination of down-sampling and up-sampling convolutional layers, and not the concatenation of the outputs of the former to the outputs of the latter.

5.6. 3D convolutional Results

Our best 3D convolutional model had a combination of concatenation and non concatenation, $3 \times 3 \times 3$ kernel sizes, kernel scalings of $[32, 64, 128, 256, 512]$, and a dropout rate of $.2$. Our best model received a dice score of $.71$ on the test set. The architecture of this model is shown in Table 1

This model had $13,522,721$ trainable parameters but it did not do significantly better than other models. We suspect that this model may have been able to find a local loss minimum among the data better than other models perhaps due to randomness in the shuffling of the dataset.

5.7. Post-Processing

We performed post-processing using an average-pooling filter with different sizes (1,8,16,32,64) and a stride of 1. We applied this filter to the binarized predicted segmentations produced by our model. After applying the filter,

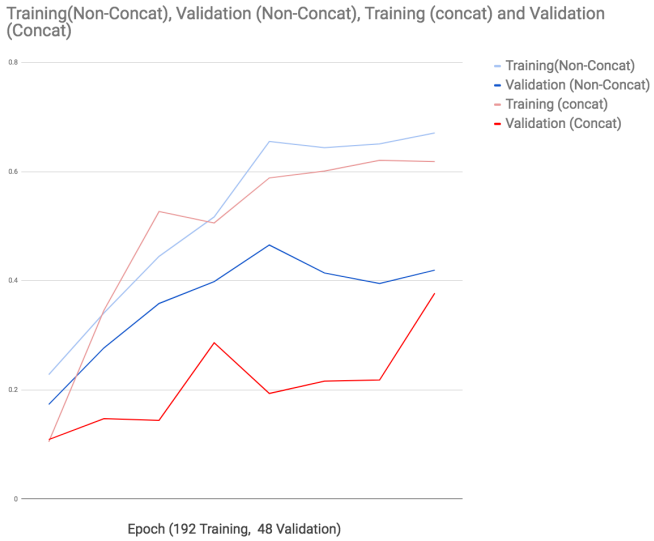

Figure 8. Training and validation curves for concatenated and non-concatenated images on the upsampling of the images in those convolutional layers. Non-concatenated imaging tended to perform better on our set. This shows training for 10 epochs.

we rounded the resulting values, so that our final segmentations were still binarized. We settled on a filter size of 16x16 after experimenting with several different sizes and comparing the trade-offs between the improved smoothness of our predicted segmentations and the increased distortion of ground-truth labels caused by larger filters. The pre-smoothing prediction and true label are shown in Figure 9. The smoothed (on the left) and the rounded (on the right) images for the 5 filters sizes are shown in 10. The best filter size seems to be 16x16.

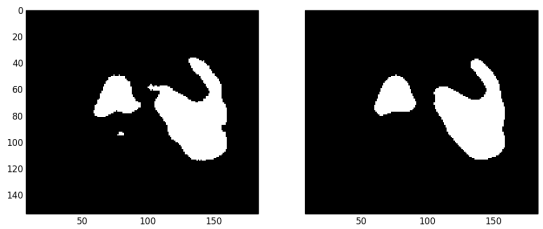

Figure 9. Example pair of an original (pre-smoothing) prediction image on the left, and the true label on the right

6. Conclusions

We have drawn several key conclusions from our experiments.

First, we found that, by slightly modifying our calculation of the Dice score coefficient, we were able to create a differentiable loss function that allowed us to directly optimize for the Dice score of our predicted segmentations.

Layer (type)	Param
conv3d_1 (Conv3D)	896
max_pooling3d_1 (MaxPooling3D)	0
dropout_1 (Dropout)	0
conv3d_2 (Conv3D)	55360
max_pooling3d_2 (MaxPooling3D)	0
dropout_2 (Dropout)	0
conv3d_3 (Conv3D)	221312
max_pooling3d_3 (MaxPooling3D)	0
dropout_3 (Dropout)	0
conv3d_4 (Conv3D)	884992
max_pooling3d_4 (MaxPooling3D)	0
dropout_4 (Dropout)	0
conv3d_5 (Conv3D)	3539456
dropout_5 (Dropout)	0
deconvolution3d_1 (Deconvolution	3539200
concatenate_1 (Concatenate 3d_1, 3d_4)	0
dropout_6 (Dropout)	0
conv3d_6 (Conv3D)	3539200
deconvolution3d_2 (Deconvolution	884864
dropout_7 (Dropout)	0
conv3d_7 (Conv3D)	442496
deconvolution3d_3 (Deconvolution	221248
dropout_8 (Dropout)	0
conv3d_8 (Conv3D)	110656
deconvolution3d_4 (Deconvolution	55328
dropout_9 (Dropout)	0
conv3d_9 (Conv3D)	27680
conv3d_10 (Conv3D)	33

Table 1. Our best 3D convolutional Model

This loss function performed significantly better than any other loss function that we tried.

Second, we discovered that 3D convolutional networks are able to perform significantly better than 2D networks on a brain tumor segmentation task. Our largest 2D network, with 21 million trainable parameters, achieved a Dice score of only 0.39 on the validation set, while we were able to achieve a validation score of nearly 0.7 using a 3D net with 1/3 the number of parameters.

Third, we determined that the concatenation of early-layer outputs to late-layer outputs is not a crucial component of the U-Net architecture. Our 3D model that used concatenation only performed slightly better on the validation set (Dice score of 0.42) than an identical model that did not use concatenation (Dice score of 0.38) after training for eight epochs.

Finally, we found that convolving with an average-pooling filter of size 16x16 with a stride of 1 serves as a simple but effective post-processing mechanism that can smooth the outputs of a semantic segmentation network.

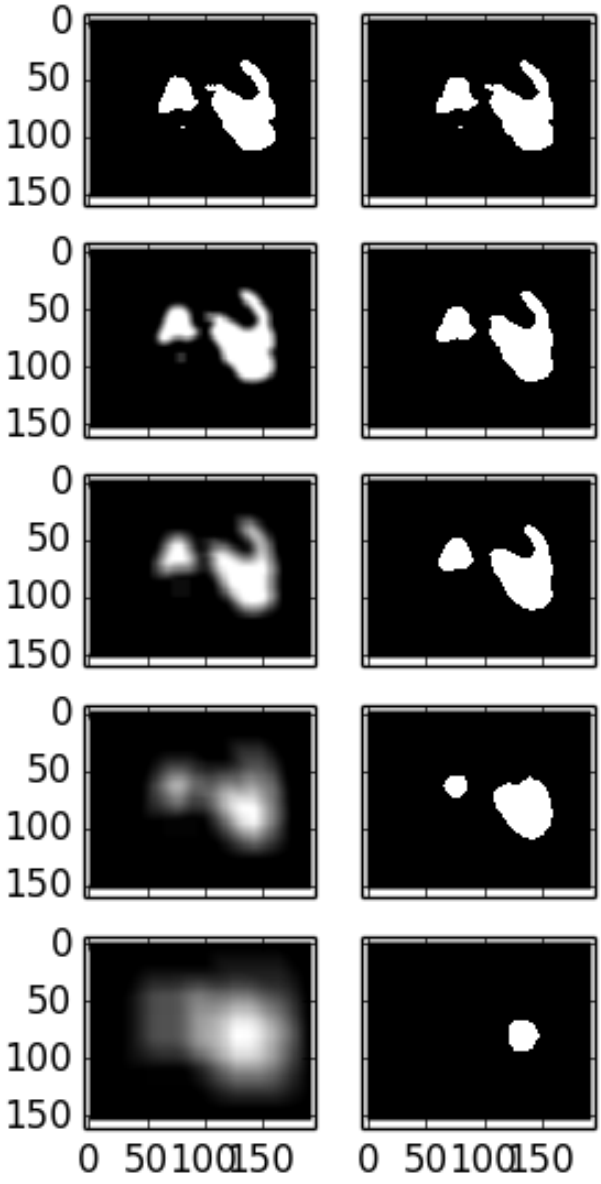

Figure 10. Varying the smoothing filter sizes. On the left the convolved image, on the right the convolved and rounded image. From top to bottom the filter sizes are 1x1, 8x8, 16x16, 32x32, 64x64

Moving forward, we would like to be able to train a significantly deeper three-dimensional network. We were unable to work with larger networks due to memory constraint issues. Assuming that we are unable to access more more powerful GPUs, we would like to experiment with Resnet-style architectural choices that would allow us to make our network much deeper without drastically increasing our memory usage. Additionally, we would like to experiment with more sophisticated post-processing techniques such as conditional random fields or sequential convolutional neural networks. We would also like to try running our 3D net on patches extracted from our brain volumes, rather than on the full volumes, so that we can train on larger batches. The benefit of training on larger batches might outweigh the cost of only allowing our filters to convolve across subsections of the brain volumes. Finally, for this project we focused only on segmenting the whole tumor. We could extend our work by taking advantage of the BraTS dataset to perform multi-class segmentation of the various tumor subsections.

References

[1] Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, and Hugo Larochelle. Brain tumor segmentation with deep neural networks. *Medical Image Analysis*, 35:18 – 31, 2017. ISSN 1361-8415. doi: https://doi.org/10.1016/j.media.2016.05.004. URL `http://www.sciencedirect.com/science/article/pii/S1361841516300330`.

[2] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(4):640–651, 2017.

[3] Alexey A Novikov, David Major, Dimitrios Lenis, Jiri Hladuvka, Maria Wimmer, and Katja Buhler. Fully convolutional architectures for multi-class segmentation in chest radiographs. *arXiv preprint arXiv:1701.08816*, 2017.

[4] Jifeng Dai, Kaiming He, Yi Li, Shaoqing Ren, and Jian Sun. Instance-sensitive fully convolutional networks. In *European Conference on Computer Vision*, pages 534–549. Springer, 2016.

[5] Konstantinos Kamnitsas, Christian Ledig, Virginia FJ Newcombe, Joanna P Simpson, Andrew D Kane, David K Menon, Daniel Rueckert, and Ben Glocker. Efficient multi-scale 3d cnn with fully connected crf for accurate brain lesion segmentation. *Medical Image Analysis*, 36:61–78, 2017.

[6] Tom Brosch, Lisa YW Tang, Youngjin Yoo, David KB Li, Anthony Traboulsee, and Roger Tam. Deep 3d convolutional encoder networks with shortcuts for multiscale feature integration applied to multiple sclerosis lesion segmentation. *IEEE transactions on medical imaging*, 35(5):1229–1239, 2016.

[7] Konstantinos Kamnitsas, Liang Chen, Christian Ledig, Daniel Rueckert, and Ben Glocker. Multi-scale 3d convolutional neural networks for lesion segmentation in brain mri. *Ischemic Stroke Lesion Segmentation*, 13, 2015.

[8] Yefeng Zheng, David Liu, Bogdan Georgescu, Hien Nguyen, and Dorin Comaniciu. 3d deep learning for efficient and robust landmark detection in volumetric data. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 565–572. Springer, 2015.

[9] Daniel Maturana and Sebastian Scherer. 3d convolutional neural networks for landing zone detection from lidar. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3471–3478. IEEE, 2015.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[11] Baris Kayalibay, Grady Jensen, and Patrick van der Smagt. Cnn-based segmentation of medical imaging data. *arXiv preprint arXiv:1701.03056*, 2017.

[12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL `http://arxiv.org/abs/1505.04597`.

[13] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[14] Matthew Lai. Deep learning for medical image segmentation. *arXiv preprint arXiv:1505.02000*, 2015.

[15] Tse-Wei Chen, Yi-Ling Chen, and Shao-Yi Chien. Fast image segmentation based on k-means clustering with histograms in hsv color space. In *Multimedia Signal Processing, 2008 IEEE 10th Workshop on*, pages 322–325. IEEE, 2008.

[16] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, L. Lanczi, E. Gerstner, M. A. Weber, T. Arbel, B. B. Avants, N. Ayache, P. Buendia, D. L. Collins, N. Cordier, J. J. Corso, A. Criminisi, T. Das, H. Delingette, . Demiralp, C. R. Durst, M. Dojat, S. Doyle, J. Festa, F. Forbes, E. Geremia, B. Glocker, P. Golland, X. Guo, A. Hamamci, K. M. Iftekharuddin, R. Jena, N. M. John, E. Konukoglu, D. Lashkari, J. A. Mariz, R. Meier, S. Pereira, D. Precup, S. J. Price, T. R. Raviv, S. M. S. Reza, M. Ryan, D. Sarikaya, L. Schwartz, H. C. Shin, J. Shotton, C. A. Silva, N. Sousa, N. K. Subbanna, G. Szekely, T. J. Taylor, O. M. Thomas, N. J. Tustison, G. Unal, F. Vasseur, M. Wintermark, D. H. Ye, L. Zhao, B. Zhao, D. Zikic, M. Prastawa, M. Reyes, and K. Van Leemput. The multimodal brain tumor image segmentation benchmark (brats). *IEEE Transactions on Medical Imaging*, 34(10):1993–2024, Oct 2015. ISSN 0278-0062. doi: 10.1109/TMI.2014.2377694.