Automatic Tumor Segmentation from MRI scans

Raunaq Rewari
Stanford University
raunaq@stanford.edu

Abstract

The motive of this paper is to come up with a fully automatic tumor segmentation approach using convolutional neural networks. Tumors can appear anywhere in the brain and have almost any kind of shape, size, and contrast. These reasons motivate the use of a flexible, high capacity deep neural network. This is a summary of the work done in this regard with an effort to describe in as much detail the methodology used. The BraTS (Brain Tumor Segmentation challenge dataset) that contains brain MRI scans for more than 200 patients, is used in this study. A patch wise segmentation approach is used and we see 93% accuracy on the test set of patches. A variety of experiments are done around the depth of the neural network used, the different architectures etc to train the best architecture for this task.

1. Introduction

The proposed networks are tailored to gliomas glioblastomas (both low and high grade) pictured in MR images. While gliomas are the most common brain tumors, they can be less aggressive (i.e. low grade) in a patient with a life expectancy of several years, or more aggressive (i.e. high grade) in a patient with a life expectancy of at most 2 years.

Although surgery is the most common treatment for brain tumors, radiation and chemotherapy may be used to slow the growth of tumors that cannot be physically removed. Magnetic resonance imaging (MRI) provides detailed images of the brain, and is one of the most common tests used to diagnose brain tumors. All the more, brain tumor segmentation from MR images can have great impact for improved diagnostics, growth rate prediction and treatment planning.

While some tumors such as meningiomas can be easily segmented, others like gliomas and glioblastomas are much more difficult to localize. These tumors (together with their surrounding edema) are often diffused, poorly contrasted, and extend tentacle-like structures that make them difficult to segment. Another fundamental difficulty with segmenting brain tumors is that they can appear anywhere in the brain, in almost any shape and size. This makes this problem a typical neural network problem.

Since glioblastomas are infiltrative tumors, their borders are often fuzzy and hard to distinguish from healthy tissues. As a solution, more than one MRI modality is often needed, e.g. T1 (spin- lattice relaxation), T1-contrasted (T1C), T2 (spin-spin relaxation), proton density (PD) contrast imaging, diffusion MRI (dMRI), and fluid attenuation inversion recovery (FLAIR) pulse sequences. The BraTS dataset provides four modalities for each patient: T1, T2, T1c and FLAIR. The contrast between these modalities gives almost a unique signature to each tissue type. We follow a patch based segmentation approach where we extract patches from the 3D image of the brain (30X30 patch size), with the label of the patch being the label of the center pixel of the patch. The idea is to use the modalities as the channels for patches of an image instead of the traditional rgb channel.

The task is not only to segment the whole glioma, but also different substructures of it. Each pixel in a volume should be classified with one label that is arranged on a scale of how "serious" the label is. The labels (shown with appropriate labels used) are, ordered after increasing seriousness:

- normal tissue (0)
- edema (1)
- non-enhancing core (2)
- necrotic core (3)
- enhancing core (4)

Hence the input to the CNN is a (4X30X30) matrix, with the goal of predicting the label of the center of the given patch. Hence, if you were to use the given architecture to segment tumors from an MRI scan, you would go from voxel to voxel, extracting a patch and predicting the label.

2. Related Work

Before the dawn of the popularity of CNNs for this task, other methods were used. Some of these methods are segmentation by thresholding[1], region based methods[2] etc. There are two kinds of general methods- generative v/s discriminative methods. The generative methods assume some prior knowledge, where methods like neural networks make no such assumption. Other neural network based approaches[3] for example predict the middle (d x d) labels in a (d' x d') [d' >d] patch. The size of d and d' has been optimized to get the best results. Other groups that have used the patch based segmentation approach use a two-pathway architecture[4], in which each pathway is responsible for learning about either the local details or the larger context of tissue appearances (e.g. whether or not it is close to the skull). The pathways are joined by concatenating their feature maps immediately before the output layer. In another method, instead of outputting labels from the CNN, probability distributions for a region are outputted which are all merged at the end to give the final segmentation. The work by Pinheiro and Collobert[6] uses a basic CNN to make predictions for each pixel and further improves the predictions by using them as extra information in the input of a second CNN model. Other work[7] involves several distinct CNNs processing the image at different resolutions. The final per-pixel class prediction is made by integrating information learned from all CNNs.

Other state of the art segmentation approaches, outside of the brain imaging domain as the fully convolutional approach for semantic segmentation[5]. This is a more computationally efficient way of performing segmentation and does not assume an input size. 3-D convolutions have also been tried for other datasets and may prove to be useful for this task.

Other segmentation tasks in the healthcare domain use the features extracted after passing through a CNN with a random forest at the end.

3. Methodology

Like mentioned earlier in the report, we use a patch based segmentation approach. The convolutional network architecture and implementation are carried out using CAFFE.

CNNs are the continuation of the multi-layer perceptron. In the MLP, a unit performs a simple computation by taking the weighted sum of all other units that serve as input to it. The network is organized into layers of units, where each unit in one layer is connected to all the other units in the previous layer.

The essence of CNNs is the convolutions. The main trick with convolutional networks that avoids the problem of too many parameters is *sparse* connections. Every unit is not connected to every other unit in the previous layer, like in traditional neural networks. Instead, every unit has its own *receptive field*, a grid of units in the previous layer which it receives input from. The receptive fields of units typically overlap. This way, the network can also take advantage of the fact that images most often have high spatially local correlations. Additionally, each unit in a layer *share* their weights. If a receptive field consists of k units, then all units in the receiving layer use the same set of k weights. However by sharing weights, only one specific feature of an image can be detected by all units in a layer. Therefore, each layer has multiple *filters*, where each filter has its own set of weights and allows for multiple features to be detected in a layer. The non-linearity generally used with CNNs is ReLU (rectified linear unit), which has the form:

$f(x) = \max(0, x)$

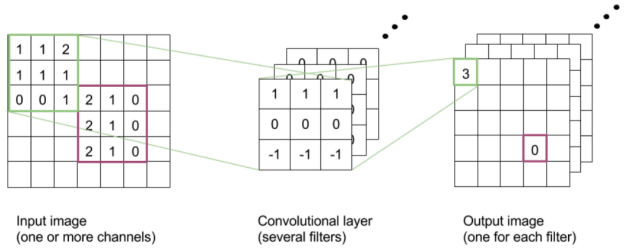

Figure 1- Figure showing a convolution layer and the workings of getting an output by convolving on the inputs.

It is generally found that increasing the number of convolutional layers (i.e. making the network deeper) results in better results by increasing the number of parameters. To prevent the model from overfitting, it is suggested to use regularization techniques like L1 or L2 regularization and dropout.

Another important concept used while designing a CNN model is *pooling*. It is used to decrease the input dimensionality of the next layer. The idea is to summarize information in a filter over a small rectangular patch of neighboring units instead of using unit output individually as input to the next layer. This method can be used between all, some or none of the convolutional layers. The main objective of performing this summary of information is to discard irrelevant details and keep the features as information-rich as possible. Examples of effects are invariance to changes in position and lighting conditions, robustness to clutter and compact representation. By reducing the input dimensionality, it also reduces the

computational costs of the network, which is necessary to implement really deep architectures. There can be multiple kinds of pooling functions possible over the neighborhood of the surrounding pixels like maximum, average of the pixels.

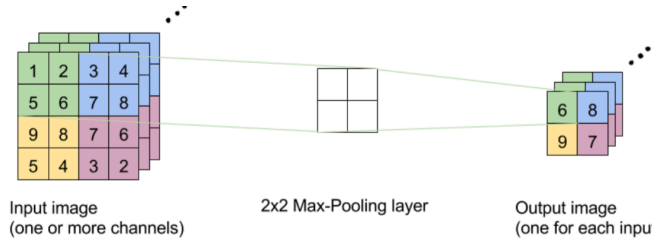

Figure 2- Figure showing an example of pooling (max pooling in this case) over a 4 x 4 neighborhood window

A typical CNN consists of convolutional layers in the beginning followed by fully-connected layers towards the end. This fully connected layer can be thought of as learning the abstract details of the image while the convolutional layers learn the local details.

The most basic piece of operation in neural networks is the $w^T x + b$ computation at each step, where w is the weight matrix and x is the input data and b is the bias. At the final layers, the same computation is used to calculate the class scores and using that the loss function is calculated. The loss function is a measure of the unhappiness of the classification. In classification, the output layer typically has a unit for each class it is supposed to classify. A simple softmax function is commonly used to make sure the output y for each position j is in the range [0,1] and sum to 1:

$$S_j(y) = \frac{e^{y_j}}{\sum_k e^{y_k}}.$$

The loss in case of softmax is the negative log of the probability of the correct class, shown as follows:

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

The more examples that the model classifies correctly, the lower the loss goes. Monitoring the loss is an excellent way of judging the efficacy of the learning system. Ideally, we want the loss to go down as much as possible.

It important to note here that the network needed to be initialized from scratch and fine-tuning of a pretrained network was not possible. This is because most pre-trained networks contain three channels, instead of the 4 in our case.

It is important to mention here that the network needs hyper-parameter tuning i.e. the model parameters like learning rate, batch size, weight decay need to be tuned.

4. Dataset

The data comes from the BRATS (multimodal **Bra**in **T**umor **S**egmentation) challenge. It consists of four MRI modalities – T1, T1-c, T2 and FLAIR. There were a total of 230 brains in the data set, out of which we only used 30.

Each entry in the dataset looks as follows:

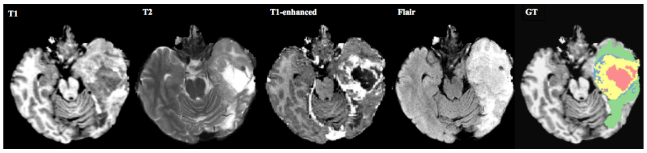

Figure 3- The first four images from left to right show the MRI modalities used as input channels to various CNN models and the fifth image shows the ground truth labels where (GREEN) edema, (YELLOW) enhanced tumor, (RED) necrosis, (BLUE) non-enhanced tumor.

Since majority of the data of the brain consists of healthy tissue (label 0), there is a need to balance the number of examples of each class. The total number of tumorous voxels is slightly more than the total number of healthy tissue voxels.

The first part of generating the data to work with is to extract 30 x 30 patches from the 3-D images by going slice by slice and extracting a 30 x 30 patch around every second voxel for the tumorous voxels and every fourth voxel from the healthy voxels. The labels of the voxel is stored separately. Both the patch and the labels are used in making lmdbs which are used to input data into CAFFE.

Each patch goes through a few pre-processing steps. The first is N4ITK bias field correction. Magnetic resonance images often exhibit image intensity non-uniformities that are the result of magnetic field variations rather than anatomical differences. These artifacts, often described as shading or bias, can be produced by imperfections in the field coils used in the systems or by magnetic susceptibility changes at the boundaries between anatomical tissue and air. To account for this variation, bias field correction is applied. Nyul's intensity normalization is applied next. Intensity normalization is an important pre-processing step in the study and analysis of Magnetic Resonance Images (MRI) of human brains. As most parametric supervised automatic image segmentation and classification methods base their assumptions regarding the intensity distributions on a standardized

intensity range, intensity normalization takes on a very significant role. One of the fast and accurate approaches proposed for intensity normalization is that of Nyul and colleagues.

The data obtained after the pre-processing is shuffled and 75% of the data is used for training, 15% for validation and the rest for training.

The expected output of the model would be as follows:

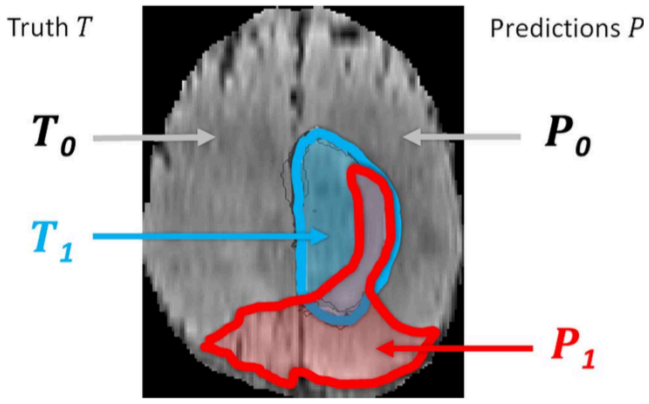

Figure 4- Region T_1 is the true lesion area (outline blue), T_0 is the remaining normal area. P_1 is the area that is predicted to be lesion by—for example—an algorithm (outlined red), and P_0 is predicted to be normal. T_1 has some overlap with P_1 in the right lateral part of the lesion, corresponding to the area referred to as T_1 ^ P_1 in the definition of the Dice score

The model will classify each voxel into one of the 5 categories and the output will be used to calculate the Dice score. For each class, we can think of outputting a binary map with algorithmic predictions P = {0,1} and the experts' consensus truth T= {0,1}, and we calculated the Dice score as:

$$\mathrm{Dice}(P,T) = \frac{|P_1 \wedge T_1|}{(|P_1| + |T_1|)/2}$$

where ^ is the logical AND operator, | . | is the size of the set (i.e., the number of voxels belonging to it). These dice scores can be compared with Dice scores obtained for doctors who have also tried to segment brain tumors from MRIs.

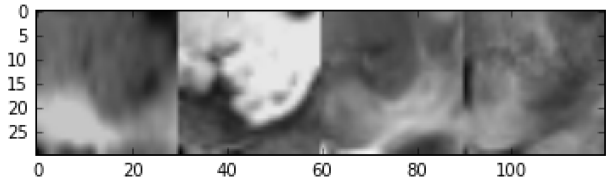

Figure 5- Figure showing the patches used as training data in the four modalities (from left to right: FLAIR, T1c, T2, T1)

5. Results and Discussion

Now that the input format of the data along with the objective is clear, it is imperative to talk about results. This part will be divided into different sections that focus on different aspects of training the best CNN. The final model that was used for training is shown below. It gives an accuracy of 93% on the test data.

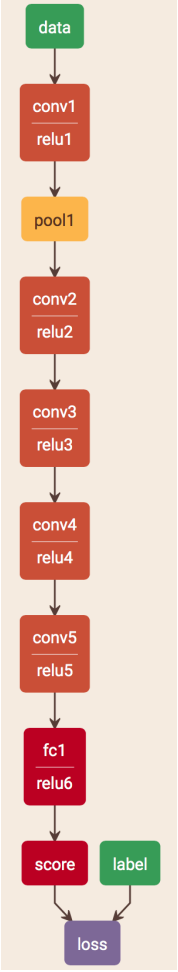

Figure 6- Final architecture used for training the model

Depth of the network

We started with a 4 layer network with 3 convolutional layers and a fully connected layer that gave an accuracy of

86% on the validation set. Increasing the number of convolutional layers till 5 increased the validation set accuracy, but an increase after that led to no change in accuracy. Even though a deeper network is generally preferred, it is always better to use one that has lower number of parameters. We also experimented with the number of fully connected layers at the end and realized that having just one was sufficient.

The size of the blobs in CAFFE that contain the data is shown below:

```
[('data', (128, 4, 30, 30)),
 ('label', (128,)),
 ('conv1', (128, 50, 30, 30)),
 ('pool1', (128, 50, 15, 15)),
 ('conv2', (128, 50, 15, 15)),
 ('conv3', (128, 50, 15, 15)),
 ('conv4', (128, 50, 15, 15)),
 ('conv5', (128, 50, 15, 15)),
 ('fc1', (128, 200)),
 ('score', (128, 5)),
 ('loss', ())]
```

The weights used with this architecture have also been outlined below:

```
[('conv1', (50, 4, 3, 3)),
 ('conv2', (50, 50, 3, 3)),
 ('conv3', (50, 50, 3, 3)),
 ('conv4', (50, 50, 3, 3)),
 ('conv5', (50, 50, 3, 3)),
 ('fc1', (200, 11250)),
 ('score', (5, 200))]
```

Number of neurons

This section involves discussion on the number of activation maps to use for the convolutional layers and the number of neurons used in the fully connected layer. The model with all activation maps as 25 gave a validation set accuracy of 75%, which prompted the increase in the number of activation maps to 50. Remember that these intermediate results being discussed here are without hyper-parameter tuning. Increasing activation maps for all layers to be 50, increased the validation set accuracy to 80%. Further increasing the activation maps lead to no substantial increase in accuracy and keeping computational costs in mind, the activation maps for all layers were fixed to 50.

Pooling layers

Like mentioned earlier, pooling is used to summarize information from the previous layer and pass it on to the next. It is used after a convolutional – ReLU layer combination. The number of pooling layers in the model were also optimized, and it was found that only one pooling layer is sufficient and the best results were seen if the pooling layer was placed immediately after the first convolutional layer. The reason behind this might be that there is too much overlap between one patch to the other and it would suit the model more if only the essential details get passed on forward.

Learning rate and solver

Various solvers for model optimization were tried in order to get the best results, like stochastic gradient descent (SGD), Adam, AdaGrad, AdaDelta and RMSProp. It was found that the Adam solver with hyper-parameter tuning gave the best results and was chose as the solver.
After some hyper-parameter tuning, the model gave the best results for a base learning rate (base_lr) of 0.001, using an “inverse” decay learning policy with gamma = 0.0001 and power = 0.75. The learning rate at the current iteration (iter) is as follows:

base_lr * (1 + gamma * iter) ^ (- power)

Batch size and epochs

The batch size refers to the number of training examples considered for one update of the optimization solver. While choosing the batch size, a couple of things are kept in mind- the computational cost and the uncertainty of update from a small batch v/s a larger batch.
The smaller batch size may give higher noise than a larger batch size. However, if the error function has a lot of local minimas, our model would get stuck in the first minima it fell into. Here, using small batches is helpful as it will get more noise in our estimate of the gradient. This noise might be enough to push us out of some of the shallow valleys in the error function. After running through a few different batch sizes (from 64 to 512), a model with batch size equal to 128 worked the best.
Also, the number of training epochs used was 2 as increasing the number of epochs after that showed no change in the accuracy.

Kernel size for convolution

Experiments with kernel size 3, 5, 7 were tried, and a decrease in accuracy was observed with increase in kernel size. This coupled with the fact that a small kernel size has lesser number of parameters presents a win - win situation and kernel size of 3 was set.

After iterating through the experiments described previously on the validation set, the final test accuracy was seen to be 93% as can be seen in figure 7.

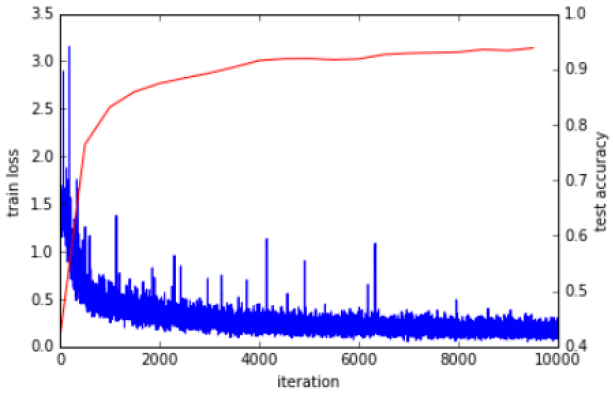

Figure 7- Figure showing the accuracy on the test set.

The confusion matrix, staple for classification problems, can also be calculated and is shown in Table 1.

Table 1- Confusion Matrix

Prediction/Truth	0	1	2	3	4
0	52205	7	746	98	47
1	30	5917	134	267	620
2	1468	109	41757	1526	485
3	51	149	638	8166	337
4	164	669	470	3	12153

Precision and recall can be calculated from the confusion matrix. Precision is the fraction of events where we *correctly* declared i out of all instances where the algorithm declared i. Conversely, recall is the fraction of events where we correctly declared i out of all of the cases where the true of state of the world is i. The precision for label 0 i.e. for the healthy tissue is 98.3%, which is a good sign as the model is incorrectly calling a patch as tumorous if its actually not.

Label wise accuracy can also be seen in Table 2.

Label	0	1	2	3	4
Accuracy	97	86	95	74	89

Table 2: Accuracy (in %)

Other architectures used

To help take into account the local and global features in the input patch, two different kernel sizes were used to train two different pathways. One method involved using a 9 x 9 kernel with pooling and convolutional layers to match the size of the data after the convolutional and pooling layers from the first path (axial size of 15 x 15). Concatenating the two paths before feeding to the fully connected layer decreased the accuracy slightly. In another architecture, a much larger 15 x 15 kernel was used with stride 15 to get a 4 x 4 output. This was combined with 15 x 15 max and average pooling on the input data to give three new pathways, which were used with the main pathway that was also altered to match the 2 x 2 output. This method very slightly increased the accuracy of the model.

6. Conclusions and future work

We were able to successfully implement a Convolutional Neural Network based approach to segment tumors from MRI scans using a moderately deep network with not too many parameters. We were able to get a high classification accuracy.

The next step on this track could be to try 3D convolutional networks on each patch and compare with the 2-D convolution method. Another possible approach could be to use a fully convolutional approach and be able to input the entire brain scan instead of patches.

8. References

1. P Gibbs, D L Buckley, S J Blackband, and A Horsman. Tumour volume determination from MR images by morphological segmentation. *Physics in medicine and biology*, 41:2437–2446, 1996.
2. YM. Salman. Modified technique for volumetric brain tumor measurements. *Journal of Biomedical Science and Engineering*, 02(February):16–19, 2009.
3. Pavel Dvorak, Bjoern Menze. Structured Prediction with Convolutional Neural Networks for Multimodal Brain Tumor Segmentation. *BRATS proceedings*
4. Mohammad Havaei, Francis Dutil, Chris Pal, Hugo Larochelle, and Pierre-Marc Jodoin A Convolutional Neural Network Approach to Brain Tumor Segmentation. *BRATS proceedings*
5. Jonathan Long, Evan Shelhamer, Trevor Darrell Fully Convolutional Networks for Semantic Segmentation
6. Pinheiro, P., Collobert, R., 2014. Recurrent convolutional neural networks for scene labeling, in: Proceedings of The 31st International Conference on Machine Learning, pp. 82–90.
7. Farabet, C., Couprie, C., Najman, L., LeCun, Y., 2013. Learning hierarchical features for scene labeling. Pattern Analysis and Machine Intelligence, IEEE Transactions on 35, 1915–1929.