Brain Tumor Segmentation with Deep Neural Networks

Axel Davy[1], Mohammad Havaei[2], David Warde-Farley[3], Antoine Biard[4], Lam Tran[5], Pierre-Marc Jodoin[2], Aaron Courville[3], Hugo Larochelle[2], Chris Pal[3,6], and Yoshua Bengio[3]

[1] École normale supérieure, Paris, France
[2] Université de Sherbrooke, Sherbrooke, Canada
[3] Université de Montréal, Montréal, Canada
[4] École polytechnique, Palaiseau, France
[5] University of Rochester, New York, USA
[6] École Polytechnique de Montréal, Canada

Abstract. Deep Neural Networks (DNNs) are often successful in problems needing to extract information from complexe, high-dimensional inputs, for which useful features are not obvious to design. This paper presents our work on applying DNNs to brain tumor segmentation for the BRATS challenge. We are currently experimenting with several several DNN architectures, leveraging the recent advances in the field such as convolutional layers, max pooling, Maxout units and Dropout regularization. We present preliminary results, for our best performing network on the BRATS2013 training set, leaderboard dataset and challenge dataset.

The results are obtained from the evaluation tool available on the Virtual Skeleton database. While we do not beat the best results of BRATS2013 participants with our current architecture, our results are promising.

1 Introduction

Deep Neural Networks (DNNs) have recently attracted more attention due to their state-of-the-art performance on several datasets such as ImageNet [7] and CIFAR-10 [5]. DNNs have also been applied successfully to segmentation problems [2, 6], the type of task considered here. However, to the best of our knowledge, there is no existing work on DNNs applied to brain tumor segmentation.

We are currently experimenting with several architectural variations of DNNs, for tackling brain segmentation. Our best architecture, which we briefly describe here, is based on convolutional layers, Maxout [5] and Dropout [9]. We also describe future variations we'd like to investigate before the end of the challenge.

The data used here is the one available for the BRATS2013 challenge, whose training set is composed of 20 brains of High Grade (HG) patients and 10 brains of Low Grade (LG) patients. There are 5 segmentation labels: Non-tumor, Necrosis, Edema, Non-enhancing tumor and Enhancing Tumor. While the BRATS2014 challenge introduces two new optional tasks (Longitudinal Lesion Segmentation and Diagnostic Image Classification), we do not plan to participate to those.

2 Methods

We start by defining some of the building blocks that we are investigating and using in our DNN architectures. Specifically, these building blocks allow us to form different types of Convolutional Neural Networks (CNNs). CNNs are a very efficient and effective class of models for computer vision, and they have been shown to learn and extract visual features able to generalize well across many tasks [3].

We attack the problem of brain tumor segmentation by solving it slice by slice from the axial view. Thus, the input x of our model corresponds to 2D image (slice), where each pixel is associated with multiple channels, each corresponding to a different image modality.

Convolutional layer CNN features are modeled by a set of kernels convolved over the input image x, followed by an optional element-wise non-linearity (e.g. a sigmoidal non-linearity). The result of the convolution of each kernel is referred to as a feature map. The size (width, height) of the kernels are hyper-parameters that must be specified by the user. However the kernel itself is learned during training. By treating the different feature maps as channels, resulting output of a convolutional layer can again be interpreted as an image, allowing for the stacking of multiple such layers.

From the neural network perspective, feature maps correspond to a layer of several hidden artificial neurons. Specifically, each coordinate within a feature map corresponds to an individual neuron, for which the size of its receptive field corresponds to the kernel's size. A kernel's value also represents the weights of the connections between the layer's neurons and the neurons in the previous layer. It is often found in practice that the learned kernels resemble edge detectors, each kernel being tuned to a different spatial frequency, scale and orientation, as is appropriate for the statistics of the training data.

Maxout convolutional layer This is a variant of a convolutional layer. In this case, each feature map is instead associated with 2 kernels. The feature map is computed by convolving both kernels and taking the pair-wise maximum value between both convolutions. See [5] for more details.

Max pooling layer In order to introduce invariance to local deformations such as translation, it has been found beneficial to subsample feature maps by taking the maximum feature (neuron) value over sub-windows, within each feature map. Such an operation is known as max pooling.

Fully connected layer Neurons in a convolutional layer have limited receptive field, meaning that each neuron only depends on a small local patch within the image. Moreover, within a feature map, neurons share the same set of weights for their connections with the previous layer. Fully connected layer do without these constraints: each hidden unit in the layer is connected to all units in the previous layer, and the weights of these connections are specific to each neuron. The size of the hidden layer must be specified and is considered as a hyper-parameter.

Fully connected Maxout layer This is simply the fully connected version of the Convolutional Maxout layer. In practice we use 5 set of weights for this layer instead of 2 as opposed to the convolutional Maxout layer.

Softmax layer This is a special case of fully connected layer, where the activation function is the softmax function: $softmax(\mathbf{a}) = \exp(\mathbf{a})/Z$ where Z is a normalization constant. In words, this function converts real valued vectors into a vector with positive entries that sum to one, and thus that can be interpreted as a probability distribution. Such a layer is usually used for the last (output) layer, to obtain a distribution over segmentation labels.

Dropout Dropout is a regularization method that stochastically adds noise in the computation of the hidden layers of a DNN. This is done by multiplying each hidden or input unit by 0 (i.e. masking) with a certain probability (e.g. 0.5), independently for each unit. This encourages the neural network to learn features that are useful "on their own" since each unit cannot assume that other units in the layer won't be masked. At test time, units are instead multiplied by one minus the probability of being masked. For more details, see [9].

The above building blocks open the door to several architectural choices in designing a DNN model. We are currently exploring several such variations. In this paper, we focus on the architecture that has been working best so far.

2.1 Preprocessing

In an attempt to test the ability of DNNs to learn useful features from scratch, we employed only minimal preprocessing. We removed the 1% highest and lowest intensities, as done in [8].

We then applied the N4ITK filter on the T1 and T1c modalities. We did not applied it to T2 and FLAIR, because the intensity of the tumor can get attenuated by the filter when the tumor region is large, especially at the center of the tumor. These choices were found to work best in our experiments. To apply N4ITK, we used ANTS [1]. We then normalized the data within each input channel, by subtracting channel's mean and dividing by the channel's standard deviation.

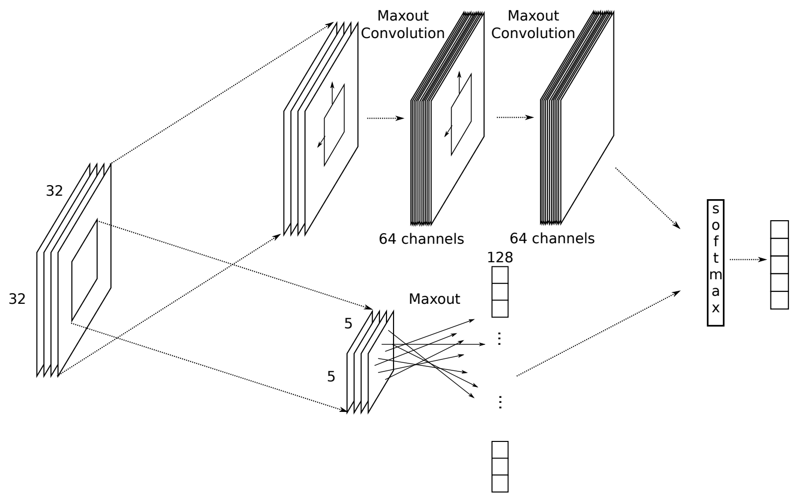

Fig. 1: Our best architecture has two paths: one concentrates on a small region around the pixel to classify, while the other looks at a wider region. The smaller path uses a fully connected Maxout layer, while the larger path is composed of two Maxout convolutional layers. The two paths' outputs are merged into a fully-connected softmax layer, which is used as our model for the segmentation's label distribution.

2.2 Current best architecture

Our best architecture is illustrated in Figure 1. It is a DNN trained on patches taken from 2D slices of the brains. Specifically, it is trained on 32x32 patches of 2D slices to predict the label of the pixel at the center of the patch.

The network has two pathways: The first is a *convolutional pathway*, connected to the entire 32x32 patch, while the second is *full-connected* to a smaller 5x5 sub-window at the center of the patch and has fewer layers. The motivation for this architectural choice is that we want the decision on the label of a pixel to be influenced by two aspects: the visual details of the region around that pixel and its larger "context" (are we near the skull, etc.). The full-connected pathway serves the first purpose while the convolutional pathway serves the latter. In our experiments, we find that the full-connected pathway is not as vital to get good performance, but helps get better contours (Figure 2).

3 Implementation details

Our implementation is based on the Pylearn2 library [4]. Pylearn2 is an open-source machine learning library specializing in deep learning algorithms. It also supports the use of GPUs, which can greatly accelerate the execution of deep learning algorithms.

To train the network, we use stochastic gradient descent with Dropout. The loss is the negative log of the probability of the correct label, where probability is read out of the output softmax layer. We first train on inputs chosen randomly, but such that all labels are equiprobable. Then, we re-train the softmax layer with a more representative distribution of the labels. We found that regularisation is very important in obtaining good results. On all the layers, we bound the absolute value of the weights and on the softmax layer we apply both L1 and L2 regularization

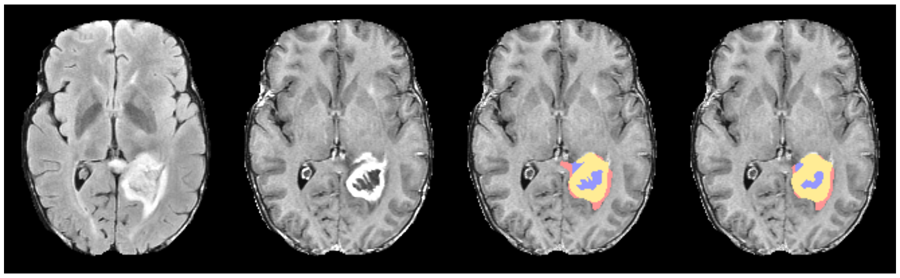

Fig. 2: The FLAIR and T1C of brain HG_0310, slice 77, followed by the segmentation produced by a network with our best architecture and by the segmentation from a similar network but without the full connected pathway. We see that the full connected pathway allows the network to more finely detail the boundary between different labels.

Name	Dice score			Positive Predictive Value			Sensitivity		
	Complete	Core	Enhancing	Complete	Core	Enhancing	Complete	Core	Enhancing
HG_0301	0.83	0.80	0.72	0.80	0.75	0.64	0.87	0.85	0.83
HG_0302	0.85	0.68	0.76	0.77	0.83	0.74	0.94	0.58	0.78
HG_0303	0.84	0.86	0.70	0.84	0.82	0.61	0.84	0.90	0.82
HG_0304	0.83	0.79	0.62	0.85	0.76	0.53	0.81	0.82	0.73
HG_0305	0.85	0.70	0.64	0.80	0.72	0.50	0.90	0.69	0.88
HG_0306	0.87	0.77	0.70	0.93	0.85	0.73	0.82	0.70	0.67
HG_0307	0.87	0.36	0.42	0.86	0.24	0.40	0.88	0.70	0.44
HG_0308	0.90	0.87	0.67	0.88	0.91	0.59	0.92	0.84	0.78
HG_0309	0.81	0.73	0.79	0.96	0.68	0.73	0.70	0.78	0.86
HG_0310	0.83	0.88	0.81	0.85	0.83	0.72	0.81	0.93	0.92
Total	0.85	0.74	0.68	0.85	0.74	0.62	0.85	0.78	0.77

Table 1: Results per brain and on the total for the 2013 Challenge dataset.

to prevent overfitting. We've also found that adding additional layers to the network doesn't give any performance improvement.

At test time, when segmenting an entire brain, we have to compute predictions one pixel at a time, which takes around 20 minutes per brain (using a GPU and including preprocessing). Faster predictions could be made by implementing the computation of both pathways as with convolutions over the entire brain. This is due to the nature of convolutions, where the weights are shared along different spatial positions.

4 Results

Table 1 shows our results on the 2013 Challenge dataset for our best architecture. We didn't have time to train and test our network on the 2014 dataset. In Table 2 are also presented our results on the Training and Leaderboard datasets.

With the current version of the architecture without any post processing, we are ranked 10^{th} on the Challenge, 8^{th} on the Training and 5^{th} on the Leaderboard datasets.

Given the minimal preprocessing we have used, these results are quite good. Additional preprocessing, such as the identification of white/gray matter and the cerebro-spinal fluid (CSF) would surely help the network have fewer false positives and increase its performance. Postprocessing could also help us remove some false positives. The network tends to have more false positives near the skull and at the top and the bottom of the brain.

4.1 Other architectural variations tested

We tried variations of the architecture to incorporate 3D information from the data. However, the results were not satisfying. A variation we tried was to give 3 adjacent patches along the third dimension as input, instead of a single slice. However it made no difference in the performance of the model, suggesting a single slice contains sufficiently enough information.

Name	Dice score			Positive Predictive Value			Sensitivity		
	Complete	Core	Enhancing	Complete	Core	Enhancing	Complete	Core	Enhancing
Train HG/LG	0.79	0.68	0.57	0.81	0.75	0.54	0.79	0.67	0.63
Leaderboad	0.72	0.63	0.56	0.69	0.64	0.50	0.82	0.68	0.68

Table 2: Results for the 2013 Training and Leaderboard datasets.

Also, we tried giving 3 orthogonal patches around the pixel to classify, by taking slides along the 3 possible directions. Due to differences in resolutions of the MRI data, we found this architecture to overfit on the training data and not generalize well.

5 Future work

We intend to further investigate architectural variations before the challenge's deadline.

Instead of training based on the prediction of an individual (center) pixel, we wish to design architectures that can jointly predict several neighbouring labels. This would allow us to more directly model the expected dependencies between the labels of nearby pixels. We mention in Section 3 that predictions at multiple locations could be obtained by implementing both the convolutional and fully-connected pathways as convolutions over larger regions than the current 32x32 input patches. We have already implemented a simpler version of this approach (without the full-connected pathway path), for which predictions on an entire brain takes around 1 minute (using a GPU). We are thus in a good position to start exploring models making structured predictions of the labels.

One approach we will investigate is to incorporate the DNN's outputs within a Conditional Random Field (CRF) model of the distribution over the labels. The CRF would incorporate pair-wise potentials between adjacent pixel positions. Another approach would be to design an architecture with cascaded predictions, where predictions further down the cascade would use as inputs the predictions computed earlier in the cascade.

6 Conclusion

In this paper we have proposed a way to do brain tumor segmentation with deep neural networks. We described our current best architecture and identified certain modeling choices that we've found important to obtain good performances. The time needed to segment an entire brain is around 20 minutes with a GPU accelerated implementation and we are confident we can decrease this to just a few minutes. We are optimistic that better results will be obtained with the not-yet-implemeted architecture using a CRF output model and improved preprocessing/postprocessing.

References

1. Brian B Avants, Nick Tustison, and Gang Song. Advanced normalization tools (ants). *Insight J*, 2009.
2. Ross Girshick Bharath Hariharan1, Pablo Arbel and Jitendra Malik. Simultaneous detection and segmentation. *arXiv preprint arXiv:1407.1808*, 2014.
3. Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In *ICML*. 2014.
4. Ian J. Goodfellow, David Warde-Farley, Pascal Lamblin, Vincent Dumoulin, Mehdi Mirza, Razvan Pascanu, James Bergstra, Frédéric Bastien, and Yoshua Bengio. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013.
5. Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *ICML*, 2013.
6. Gary B. Huang and Viren Jain. Deep and wide multiscale recursive networks for robust image labeling. *arXiv preprint arXiv:1310.0354*, 2013.
7. A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*. 2012.
8. Chris Durst Nick Tustison, Max Wintermark and Brian Avants. Ants and árboles. In *NCI-MICCAI BRATS*, 2013.
9. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.