Medical Image Analysis 35 (2017) 18–31

Contents lists available at ScienceDirect

Medical Image Analysis

journal homepage: www.elsevier.com/locate/media

Brain tumor segmentation with Deep Neural Networks

Mohammad Havaei[a,*], Axel Davy[b], David Warde-Farley[c], Antoine Biard[c,d], Aaron Courville[c], Yoshua Bengio[c], Chris Pal[c,e], Pierre-Marc Jodoin[a], Hugo Larochelle[a]

[a] Université de Sherbrooke, Sherbrooke, Qc, Canada
[b] École Normale supérieure, Paris, France
[c] Université de Montréal, Montréal, Canada
[d] École polytechnique, Palaiseau, France
[e] École Polytechnique de Montréal, Canada

ARTICLE INFO

Article history:
Received 27 April 2015
Revised 2 March 2016
Accepted 11 May 2016
Available online 19 May 2016

Keywords:
Brain tumor segmentation
Deep neural networks
Convolutional neural networks
Cascaded convolutional neural networks

ABSTRACT

In this paper, we present a fully automatic brain tumor segmentation method based on Deep Neural Networks (DNNs). The proposed networks are tailored to glioblastomas (both low and high grade) pictured in MR images. By their very nature, these tumors can appear anywhere in the brain and have almost any kind of shape, size, and contrast. These reasons motivate our exploration of a machine learning solution that exploits a flexible, high capacity DNN while being extremely efficient. Here, we give a description of different model choices that we've found to be necessary for obtaining competitive performance. We explore in particular different architectures based on Convolutional Neural Networks (CNN), i.e. DNNs specifically adapted to image data.

We present a novel CNN architecture which differs from those traditionally used in computer vision. Our CNN exploits both local features as well as more global contextual features simultaneously. Also, different from most traditional uses of CNNs, our networks use a final layer that is a convolutional implementation of a fully connected layer which allows a 40 fold speed up. We also describe a 2-phase training procedure that allows us to tackle difficulties related to the imbalance of tumor labels. Finally, we explore a cascade architecture in which the output of a basic CNN is treated as an additional source of information for a subsequent CNN. Results reported on the 2013 BRATS test data-set reveal that our architecture improves over the currently published state-of-the-art while being over 30 times faster.

1. Introduction

In the United States alone, it is estimated that 23,000 new cases of brain cancer will be diagnosed in 2015.[1] 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. Furthermore, unlike images derived from X-ray computed tomography (CT) scans, the scale of voxel values in MR images is not standardized. Depending on the type of MR machine used (1.5, 3 or 7 tesla) and the acquisition protocol (field of view value, voxel resolution, gradient strength, b0 value, etc.), the same tumorous cells may end up having drastically different gray-scale values when pictured in different hospitals.

Healthy brains are typically made of 3 types of tissues: the white matter, the gray matter, and the cerebrospinal fluid. The goal of brain tumor segmentation is to detect the location and

* Corresponding author.
E-mail address: seyed.mohammad.havaei@usherbrooke.ca (M. Havaei).
[1] http://www.ucancer.org

http://dx.doi.org/10.1016/j.media.2016.05.004
1361-8415/

extension of the tumor regions, namely active tumorous tissue (vascularized or not), necrotic tissue, and edema (swelling near the tumor). This is done by identifying abnormal areas when compared to normal tissue. 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 employed, 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 contrast between these modalities gives almost a unique signature to each tissue type.

Most automatic brain tumor segmentation methods use hand-designed features (Farahani et al., 2014; Menze et al., 2014). These methods implement a classical machine learning pipeline according to which features are first extracted and then given to a classifier whose training procedure does not affect the nature of those features. An alternative approach for designing task-adapted feature representations is to *learn* a hierarchy of increasingly complex features directly from in-domain data. Deep neural networks have been shown to excel at learning such feature hierarchies (Bengio et al., 2013). In this work, we apply this approach to learn feature hierarchies adapted specifically to the task of brain tumor segmentation that combine information across MRI modalities.

Specifically, we investigate several choices for training CNNs, which are DNNs adapted to image data. We report their advantages, disadvantages and performance using well established metrics. Although CNNs first appeared over two decades ago (LeCun et al., 1998), they have recently become a mainstay of the computer vision community due to their record-shattering performance in the ImageNet Large-Scale Visual Recognition Challenge (Krizhevsky et al., 2012). While CNNs have also been successfully applied to segmentation problems (Alvarez et al., 2012; Long et al., 2015; Hariharan et al., 2014; Ciresan et al., 2012), most of the previous work has focused on non-medical tasks and many involve architectures that are not well suited to medical imagery or brain tumor segmentation in particular. Our preliminary work on using convolutional neural networks for brain tumor segmentation together with two other methods using CNNs was presented in BRATS'14 workshop. However, those results were incomplete and required more investigation (More on this in Section 2).

In this paper, we propose a number of specific CNN architectures for tackling brain tumor segmentation. Our architectures exploit the most recent advances in CNN design and training techniques, such as Max-out (Goodfellow et al., 2013b) hidden units and Dropout (Srivastava et al., 2014) regularization. We also investigate several architectures which take into account both the local shape of tumors as well as their context.

One problem with many machine learning methods is that they perform pixel classification without taking into account the local dependencies of labels (i.e. segmentation labels are conditionally independent given the input image). To account for this, one can employ structured output methods such as conditional random fields (CRFs), for which inference can be computationally expensive. Alternatively, one can model label dependencies by considering the pixel-wise probability estimates of an initial CNN as additional input to certain layers of a second DNN, forming a cascaded architecture. Since convolutions are efficient operations, this approach can be significantly faster than implementing a CRF.

We focus our experimental analysis on the fully-annotated MICCAI brain tumor segmentation (BRATS) challenge 2013 data-set (Farahani et al., 2014) using the well defined training and testing splits, thereby allowing us to compare directly and quantitatively to a wide variety of other methods.

Our contributions in this work are four fold:

1. We propose a fully automatic method with results currently ranked second on the BRATS 2013 scoreboard;
2. To segment a brain, our method takes between 25 s and 3 min, which is one order of magnitude faster than most state-of-the-art methods.
3. Our CNN implements a novel two-pathway architecture that learns about the local details of the brain as well as the larger context. We also propose a two-phase training procedure which we have found is critical to deal with imbalanced label distributions. Details of these contributions are described in Sections 3.1.1 and 3.2.
4. We employ a novel cascaded architecture as an efficient and conceptually clean alternative to popular structured output methods. Details on those models are presented in Section 3.1.2.

2. Related work

As noted by Menze et al. (2014), the number of publications devoted to automated brain tumor segmentation has grown exponentially in the last several decades. This observation not only underlines the need for automatic brain tumor segmentation tools, but also shows that research in that area is still a work in progress.

Brain tumor segmentation methods (especially those devoted to MRI) can be roughly divided in two categories: those based on generative models and those based on discriminative models (Menze et al., 2014; Bauer et al., 2013; Angelini et al., 2007).

Generative models rely heavily on domain-specific prior knowledge about the appearance of both healthy and tumorous tissues. Tissue appearance is challenging to characterize, and existing generative models usually identify a tumor as being a shape or a signal which deviates from a normal (or average) brain (Clark et al., 1998). Typically, these methods rely on anatomical models obtained after aligning the 3D MR image on an atlas or a template computed from several healthy brains (Doyle et al., 2013). A typical generative model of MR brain images can be found in Prastawa et al. (2004). Given the ICBM brain atlas, the method aligns the brain to the atlas and computes posterior probabilities of healthy tissues (white matter, gray matter and cerebrospinal fluid). Tumorous regions are then found by localizing voxels whose posterior probability is below a certain threshold. A post-processing step is then applied to ensure good spatial regularity. Prastawa et al. (2003), also register brain images onto an atlas in order to get a probability map for abnormalities. An active contour is then initialized on this map and iterated until the change in posterior probability is below a certain threshold. Many other active-contour methods along the same lines have been proposed (Khotanlou et al., 2009; Cobzas et al., 2007; Popuri et al., 2012), all of which depend on left-right brain symmetry features and/or alignment-based features. Note that since aligning a brain with a large tumor onto a template can be challenging, some methods perform registration and tumor segmentation at the same time (Kwon et al., 2014; Parisot et al., 2012).

Other approaches for brain tumor segmentation employ discriminative models. Unlike generative modeling approaches, these approaches exploit little prior knowledge on the brain's anatomy and instead rely mostly on the extraction of [a large number of] low level image features, directly modeling the relationship between these features and the label of a given voxel. These features may be raw input pixels values (Havaei et al., 2014; Hamamci et al., 2012), local histograms (Kleesiek et al., 2014; R.Meier et al., 2014) texture features such as Gabor filterbanks (Subbanna et al., 2013; 2014), or alignment-based features such as inter-image gradient, region shape difference, and symmetry analysis (N.Tustison and Avants, 2013). Classical discriminative learning techniques such as SVMs (Bauer et al., 2011; Schmidt et al., 2005; Lee et al., 2005) and decision forests (Zikic et al., 2012) have also been used. Results

from the 2012, 2013 and 2014 editions of the MICCAI-BRATS Challenge suggest that methods relying on random forests are among the most accurate (Menze et al., 2014; Gotz et al., 2014; Kleesiek et al., 2014).

One common aspect with discriminative models is their implementation of a conventional machine learning pipeline relying on hand-designed features. For these methods, the classifier is trained to separate healthy from non-healthy tissues assuming that the input features have a sufficiently high discriminative power since the behavior the classifier is independent from nature of those features. One difficulty with methods based on hand-designed features is that they often require the computation of a large number of features in order to be accurate when used with many traditional machine learning techniques. This can make them slow to compute and expensive memory-wise. More efficient techniques employ lower numbers of features, using dimensionality reduction or feature selection methods, but the reduction in the number of features is often at the cost of reduced accuracy.

By their nature, many hand-engineered features exploit very generic edge-related information, with no specific adaptation to the domain of brain tumors. Ideally, one would like to have features that are composed and refined into higher-level, task-adapted representations. Recently, preliminary investigations have shown that the use of deep CNNs for brain tumor segmentation makes for a very promising approach (see the BRATS 2014 challenge workshop papers of Davy et al. (2014); Zikic et al. (2014); Urban et al. (2014)). All three methods divide the 3D MR images into 2D (Davy et al., 2014; Zikic et al., 2014) or 3D patches (Urban et al., 2014) and train a CNN to predict its center pixel class. Urban et al. (2014) as well as Zikic et al. (2014) implemented a fairly common CNN, consisting of a series of convolutional layers, a non-linear activation function between each layer and a soft-max output layer. Our work here[2] extends our preliminary results presented in Davy et al. (2014), using a two-pathway architecture, which we use here as a building block.

In computer vision, CNN-based segmentation models have typically been applied to natural scene labeling. For these tasks, the inputs to the model are the RGB channels of a patch from a color image. The work in Pinheiro and Collobert (2014) 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 (Farabet et al., 2013) 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. To produce a smooth segmentation, these predictions are regularized using a more global super-pixel segmentation of the image. Like our work, other recent work has exploited convolution operations in the final layer of a network to extend traditional CNN architectures for semantic scene segmentation (Long et al., 2015). In the medical imaging domain in general there has been comparatively less work using CNNs for segmentation. However, some notable recent work by Huang and Jain (2013) has used CNNs to predict the boundaries of neural tissue in electron microscopy images. Here we explore an approach with similarities to the various approaches discussed above, but in the context of brain tumor segmentation.

3. Our Convolutional Neural Network approach

Since the brains in the BRATS data-set lack resolution in the third dimension, we consider performing the segmentation slice by slice from the axial view. Thus, our model processes sequentially each 2D axial image (slice) where each pixel is associated with different image modalities namely; T1, T2, T1C and FLAIR. Like most CNN-based segmentation models (Pinheiro and Collobert, 2014; Farabet et al., 2013), our method predicts the class of a pixel by processing the $M \times M$ patch centered on that pixel. The input $\mathbf{X}$ of our CNN model is thus an $M \times M$ 2D patch with several modalities.

The main building block used to construct a CNN architecture is the *convolutional layer*. Several layers can be stacked on top of each other forming a hierarchy of features. Each layer can be understood as extracting features from its preceding layer into the hierarchy to which it is connected. A single convolutional layer takes as input a stack of input planes and produces as output some number of output planes or *feature maps*. Each feature map can be thought of as a topologically arranged map of responses of a particular spatially local non-linear feature extractor (the parameters of which are learned), applied identically to each spatial neighborhood of the input planes in a sliding window fashion. In the case of a first convolutional layer, the individual input planes correspond to different MRI modalities (in typical computer vision applications, the individual input planes correspond to the red, green and blue color channels). In subsequent layers, the input planes typically consist of the feature maps of the previous layer.

Computing a feature map in a convolutional layer (see Fig. 1) consists of the following three steps:

1. *Convolution of kernels (filters):* Each feature map $\mathbf{O}_s$ is associated with one kernel (or several, in the case of Max-out). The feature map $\mathbf{O}_s$ is computed as follows:

$$\mathbf{O}_s = b_s + \sum_r \mathbf{W}_{sr} * \mathbf{X}_r \tag{1}$$

where $\mathbf{X}_r$ is the rth input channel, $\mathbf{W}_{sr}$ is the sub-kernel for that channel, $*$ is the convolution operation and b_s is a bias term.[3] In other words, the affine operation being performed for each feature map is the *sum* of the application of R different 2-dimensional $N \times N$ convolution filters (one per input channel/modality), plus a bias term which is added pixel-wise to each resulting spatial position. Though the input to this operation is a $M \times M \times R$ 3-dimensional tensor, the spatial topology being considered is 2-dimensional in the X-Y axial plane of the original brain volume.

Whereas traditional image feature extraction methods rely on a fixed recipe (sometimes taking the form of convolution with a linear e.g. Gabor filter bank), the key to the success of convolutional neural networks is their ability to learn the weights and biases of individual feature maps, giving rise to data-driven, customized, task-specific dense feature extractors. These parameters are adapted via stochastic gradient descent on a surrogate loss function related to the misclassification error, with gradients computed efficiently via the back-propagation algorithm (Rumelhart et al., 1988).

Special attention must be paid to the treatment of border pixels by the convolution operation. Throughout our architecture, we employ the so-called *valid-mode* convolution, meaning that the filter response is not computed for pixel positions that are less than $\lfloor N/2 \rfloor$ pixels away from the image border. An $N \times N$ filter convolved with an $M \times M$ input patch will result in a $Q \times Q$ output, where $Q = M - N + 1$. In Fig. 1, $M = 7$, $N = 3$ and thus $Q = 5$. Note that the size (spatial width and height) of the kernels are hyper-parameters that must be specified by the user.

2. *Non-linear activation function:* To obtain features that are non-linear transformations of the input, an element-wise non-

[2] It is important to note that while we did participate in the BRATS 2014 challenge, we could not report complete and fair experiments for it at the time of submitting this manuscript. See Section 5 for a discussion on this point.

[3] Since the convolutional layer is associated to R input channels, $\mathbf{X}$ contains $M \times M \times R$ gray-scale values and thus each kernel $\mathbf{W}_s$ contains $N \times N \times R$ weights. Accordingly, the number of parameters in a convolutional block of consisting of S feature maps is equal to $R \times M \times M \times S$.

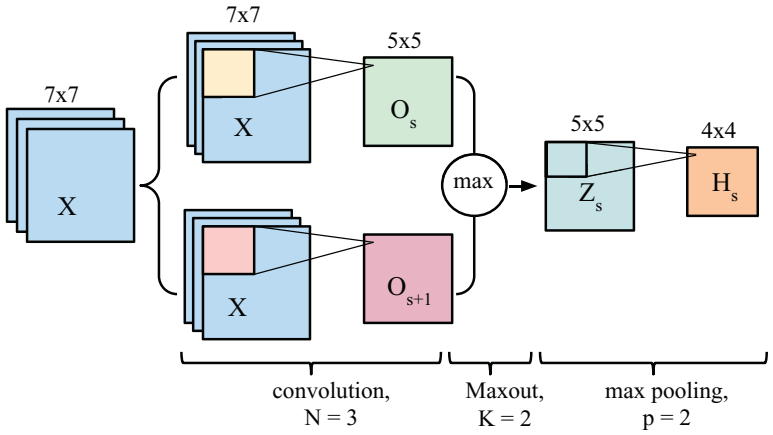


Fig. 1. A single convolution layer block showing computations for a single feature map. The input patch (here 7 × 7), is convolved with series of kernels (here 3 × 3) followed by Max-out and max-pooling.

linearity is applied to the result of the kernel convolution. There are multiple choices for this non-linearity, such as the sigmoid, hyperbolic tangent and rectified linear functions (Jarrett et al., 2009), (Glorot et al., 2011).

Recently, Goodfellow et al. (2013b), proposed a Max-out non-linearity, which has been shown to be particularly effective at modeling useful features. Max-out features are associated with multiple kernels $\mathbf{W}_s$. This implies each Max-out map $\mathbf{Z}_s$ is associated with K feature maps : $\{\mathbf{O}_s, \mathbf{O}_{s+1}, ..., \mathbf{O}_{s+K-1}\}$. Note that in Fig. 1, the Max-out maps are associated with $K = 2$ feature maps. Max-out features correspond to taking the max over the feature maps $\mathbf{O}$, individually for each spatial position:

$$Z_{s,i,j} = \max\{O_{s,i,j}, O_{s+1,i,j}, ..., O_{s+K-1,i,j}\} \quad (2)$$

where i, j are spatial positions. Max-out features are thus equivalent to using a convex activation function, but whose shape is adaptive and depends on the values taken by the kernels.

3. *Max pooling:* This operation consists of taking the maximum feature (neuron) value over sub-windows within each feature map. This can be formalized as follows:

$$H_{s,i,j} = \max_p Z_{s,i+p,j+p}, \quad (3)$$

where p determines the max pooling window size. The sub-windows can be overlapping or not (Fig. 1 shows an overlapping configuration). The max-pooling operation shrinks the size of the feature map. This is controlled by the pooling size p and the stride hyper-parameter, which corresponds to the horizontal and vertical increments at which pooling sub-windows are positioned. Let S be the stride value and $Q \times Q$ be the shape of the feature map before max-pooling. The output of the max-pooling operation would be of size $D \times D$, where $D = (Q - p)/S + 1$. In Fig. 1, since $Q = 5, p = 2, S = 1$, the max-pooling operation results into a $D = 4$ output feature map. The motivation for this operation is to introduce invariance to local translations. This sub-sampling procedure has been found beneficial in other applications (Krizhevsky et al., 2012).

Convolutional networks have the ability to extract a hierarchy of increasingly complex features which makes them very appealing. This is done by treating the output feature maps of a convolutional layer as input channels to the subsequent convolutional layer.

From the neural network perspective, feature maps correspond to a layer of hidden units or 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.

Finally, to perform a prediction of the segmentation labels, we connect the last convolutional hidden layer to a convolutional output layer followed by a non-linearity (i.e. no pooling is performed). It is necessary to note that, for segmentation purposes, a conventional CNN will not yield an efficient test time since the output layer is typically fully connected. By using a convolution at the end, for which we have an efficient implementation, the prediction at test time for a whole brain will be 45 times faster. The convolution uses as many kernels as there are different segmentation labels (in our case five). Each kernel thus acts as the ultimate detector of tissue from one of the segmentation labels. We use the *softmax* non-linearity which normalizes the result of the kernel convolutions into a multinominal distribution over the labels. Specifically, let $\mathbf{a}$ be the vector of values at a given spatial position, it computes $\text{softmax}(\mathbf{a}) = \exp(\mathbf{a})/Z$ where $Z = \sum_c \exp(a_c)$ is a normalization constant. More details will be discussed in Section 4.

Noting $\mathbf{Y}$ as the segmentation label field over the input patch $\mathbf{X}$, we can thus interpret each spatial position of the convolutional output layer as providing a model for the likelihood distribution $p(Y_{ij}|\mathbf{X})$, where Y_{ij} is the label at position i, j. We get the probability of all labels simply by taking the product of each conditional $p(\mathbf{Y}|\mathbf{X}) = \prod_{ij} p(Y_{ij}|\mathbf{X})$. Our approach thus performs a multi-class labeling by assigning to each pixel the label with the largest probability.

3.1. The architectures

Our description of CNNs so far suggests a simple architecture corresponding to a single stack of several convolutional layers. This configuration is the most commonly implemented architecture in the computer vision literature. However, one could imagine other architectures that might be more appropriate for the task at hand.

In this work, we explore a variety of architectures by using the concatenation of feature maps from different layers as another operation when composing CNNs. This operation allows us to construct architectures with multiple computational paths, which can each serve a different purpose. We now describe the two types of architectures that we explore in this work.

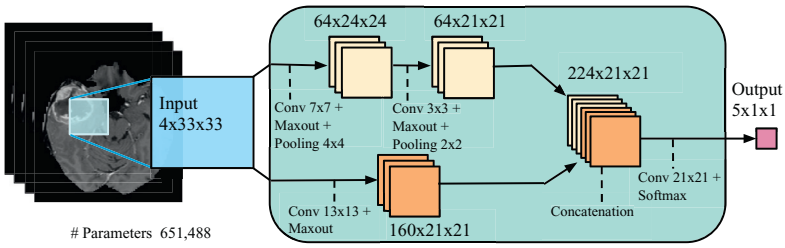


Fig. 2. Two-pathway CNN architecture (TwoPathCNN). The figure shows the input patch going through two paths of convolutional operations. The feature-maps in the local and global paths are shown in yellow and orange respectively. The convolutional layers used to produce these feature-maps are indicated by dashed lines in the figure. The green box embodies the whole model which in later architectures will be used to indicate the TwoPathCNN.

3.1.1. Two-pathway architecture

This architecture is made of two streams: a pathway with smaller 7×7 receptive fields and another with larger 13×13 receptive fields. We refer to these streams as the *local* pathway and the *global* pathway, respectively. The motivation for this architectural choice is that we would like the prediction of the label of a pixel to be influenced by two aspects: the visual details of the region around that pixel and its larger "context", i.e. roughly where the patch is in the brain.

The full architecture along with its details is illustrated in Fig. 2. We refer to this architecture as the TwoPathCNN. To allow for the concatenation of the top hidden layers of both pathways, we use two layers for the local pathway, with 3×3 kernels for the second layer. While this implies that the effective receptive field of features in the top layer of each pathway is the same, the global pathway's parametrization more directly and flexibly models features in that same area. The concatenation of the feature maps of both pathways is then fed to the output layer.

3.1.2. Cascaded architectures

One disadvantage of the CNNs described so far is that they predict each segmentation label separately from each other. This is unlike a large number of segmentation methods in the literature, which often propose a joint model of the segmentation labels, effectively modeling the direct dependencies between spatially close labels. One approach is to define a conditional random field (CRF) over the labels and perform mean-field message passing inference to produce a complete segmentation. In this case, the final label at a given position is effectively influenced by the models beliefs about what the label is in the vicinity of that position.

On the other hand, inference in such joint segmentation methods is typically more computationally expensive than a simple feed-forward pass through a CNN. This is an important aspect that one should take into account if automatic brain tumor segmentation is to be used in a day-to-day practice.

Here, we describe CNN architectures that both exploit the efficiency of CNNs, while also more directly model the dependencies between adjacent labels in the segmentation. The idea is simple: since we'd like the ultimate prediction to be influenced by the model's beliefs about the value of nearby labels, we propose to feed the output probabilities of a first CNN as additional inputs to the layers of a second CNN. Again, we do this by relying on the concatenation of convolutional layers. In this case, we simply concatenate the output layer of the first CNN with any of the layers in the second CNN. Moreover, we use the same two-pathway structure for both CNNs. This effectively corresponds to a cascade of two CNNs, thus we refer to such models as cascaded architectures.

In this work, we investigated three cascaded architectures that concatenate the first CNN's output at different levels of the second CNN:

- *Input concatenation:* In this architecture, we provide the first CNN's output directly as input to the second CNN. They are thus simply treated as additional image channels of the input patch. The details are illustrated in Fig. 3a. We refer to this model as InputCascadeCNN.
- *Local pathway concatenation:* In this architecture, we move up one layer in the local pathway and perform concatenation to its first hidden layer, in the second CNN. The details are illustrated in Fig. 3b. We refer to this model as LocalCascadeCNN.
- *Pre-output concatenation:* In this last architecture, we move to the very end of the second CNN and perform concatenation right before its output layer. This architecture is interesting, as it is similar to the computations made by one pass of mean-field inference (Xing et al., 2002) in a CRF whose pairwise potential functions are the weights in the output kernels. From this view, the output of the first CNN is the first iteration of mean-field, while the output of the second CNN would be the second iteration. The difference with regular mean-field however is that our CNN allows the output at one position to be influenced by its previous value, and the convolutional kernels are not the same in the first and second CNN. The details are illustrated in Fig. 3c. We refer to this model as MFCascadeCNN.

3.2. Training

3.2.0.1. Gradient descent. By interpreting the output of the convolutional network as a model for the distribution over segmentation labels, a natural training criteria is to maximize the probability of all labels in our training set or, equivalently, to minimize the negative log-probability $-\log p(\mathbf{Y}|\mathbf{X}) = \sum_{ij} -\log p(Y_{ij}|\mathbf{X})$ for each labeled brain.

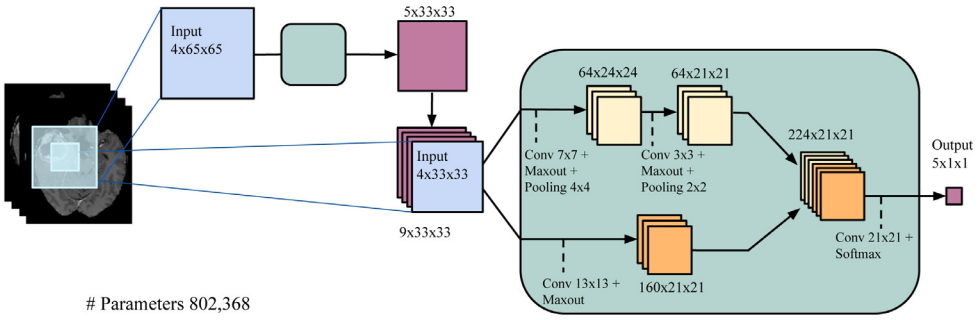
To do this, we follow a stochastic gradient descent approach by repeatedly selecting labels Y_{ij} at a random subset of patches within each brain, computing the average negative log-probabilities for this mini-batch of patches and performing a gradient descent step on the CNNs parameters (i.e. the kernels at all layers).

Performing updates based only on a small subset of patches allows us to avoid having to process a whole brain for each update, while providing reliable enough updates for learning. In practice, we implement this approach by creating a data-set of mini-batches of smaller brain image patches, paired with the corresponding center segmentation label as the target.

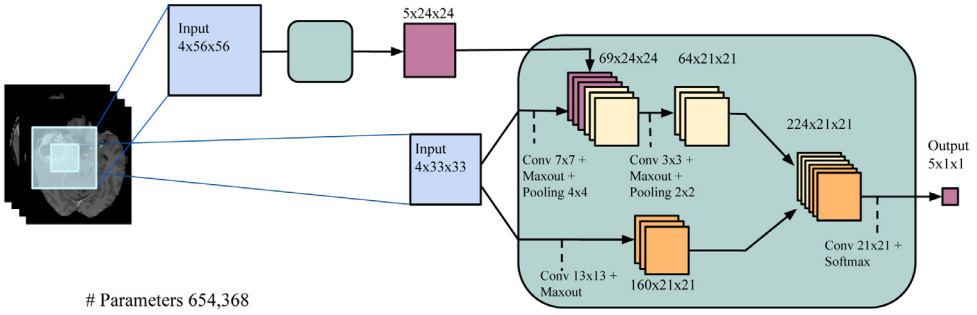
To further improve optimization, we implemented a so-called *momentum* strategy which has been shown successful in the past (Krizhevsky et al., 2012). The idea of momentum is to use a temporally averaged gradient in order to damp the optimization velocity:

$$\mathbf{V}_{i+1} = \mu * \mathbf{V}_i - \alpha * \nabla \mathbf{W}_i$$
$$\mathbf{W}_{i+1} = \mathbf{W}_i + \mathbf{V}_{i+1}$$

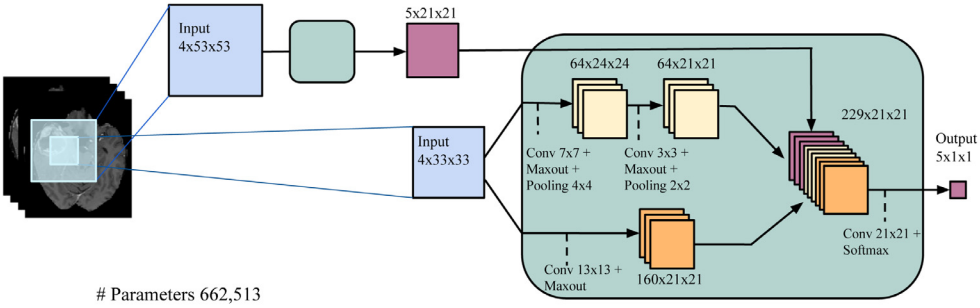
where $\mathbf{W}_i$ stands for the CNNs parameters at iteration i, $\nabla \mathbf{W}_i$ the gradient of the loss function at $\mathbf{W}_i$, $\mathbf{V}$ is the integrated velocity initialized at zero, α is the learning rate, and μ the momentum coefficient. We define a schedule for the momentum μ where



(a) Cascaded architecture, using input concatenation (InputCascadeCNN).



(b) Cascaded architecture, using local pathway concatenation (LocalCascadeCNN).



(c) Cascaded architecture, using pre-output concatenation, which is an architecture with properties similar to that of learning using a limited number of mean-field inference iterations in a CRF (MFCascadeCNN).

Fig. 3. Cascaded architectures.

the momentum coefficient is gradually increased during training. In our experiments the initial momentum coefficient was set to $\mu = 0.5$ and the final value was set to $\mu = 0.9$.

Also, the learning rate α is decreased by a factor at every epoch. The initial learning rate was set to $\alpha = 0.005$ and the decay factor to 10^{-1}.

3.2.0.2. Two-phase training. Brain tumor segmentation is a highly data imbalanced problem where the healthy voxels (i.e. label 0) comprise 98% of total voxels. From the remaining 2% pathological voxels, 0.18% belongs to necrosis (label 1), 1.1% to edema (label 2), 0.12% to non-enhanced (label 3) and 0.38% to enhanced tumor (label 4). Selecting patches from the true distribution would cause the model to be overwhelmed by healthy patches and causing problem when training out CNN models. Instead, we initially construct our patches data-set such that all labels are equiprobable. This is what we call the *first* training phase. Then, in a *second* phase, we account for the un-balanced nature of the data and re-train only the output layer (i.e. keeping the kernels of all other layers fixed) with a more representative distribution of the labels. This way we get the best of both worlds: most of the capacity (the lower layers) is used in a balanced way to account for the diversity in all of the classes, while the output probabilities are calibrated correctly (thanks to the re-training of the output layer with the natural frequencies of classes in the data).

3.2.0.3. Regularization. Successful CNNs tend to be models with a lot of capacity, making them vulnerable to overfitting in a setting like ours where there clearly are not enough training examples. Accordingly, we found that regularization is important in obtaining good results. Here, regularization took several forms. First, in all layers, we bounded the absolute value of the kernel weights and applied both L1 and L2 regularization to prevent overfitting. This is done by adding the regularization terms to the negative log-

probability (i.e. $-\log p(\mathbf{Y}|\mathbf{X}) + \lambda_1\|\mathbf{W}\|_1 + \lambda_2\|\mathbf{W}\|^2$, where λ_1 and λ_2 are coefficients for L1 and L2 regularization terms respectively). L1 and L2 affect the parameters of the model in different ways, while L1 encourages sparsity, L2 encourages small values. We also used a validation set for early stopping, i.e. stop training when the validation performance stopped improving. The validation set was also used to tune the other hyper-parameters of the model. The reader shall note that the hyper-parameters of the model which includes using or not L2 and/or L1 coefficients were selected by doing a grid search over range of parameters. The chosen hyper-parameters were the ones for which the model performed best on a validation set.

Moreover, we used *Dropout* (Srivastava et al., 2014), a recent regularization method that works by stochastically adding noise in the computation of the hidden layers of the CNN. 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 and training update. This encourages the neural network to learn features that are useful "on their own", since each unit cannot assume that other units in the same layer won't be masked as well and co-adapt its behavior. At test time, units are instead multiplied by one minus the probability of being masked. For more details, see Srivastava et al. (2014).

Considering the large number of parameters our model has, one might think that even with our regularization strategy, the 30 training brains from BRATS 2013 are too few to prevent overfitting. But as will be shown in the results section, our model generalizes well and thus do not overfit. One reason for this is the fact that each brain comes with 200 2d slices and thus, our model has approximately 6000 2D images to train on. We shall also mention that by their very nature, MRI images of brains are very similar from one patient to another. Since the variety of those images is much lower than those in real-image data-sets such as CIFAR and ImageNet, a fewer number of training samples is thus needed.

3.2.0.4. Cascaded architectures. To train a cascaded architecture, we start by training the TwoPathCNN with the two phase stochastic gradient descent procedure described previously. Then, we fix the parameters of the TwoPathCNN and include it in the cascaded architecture (be it the InputCascadeCNN, the LocalCascadeCNN, or the MFCascadeCNN) and move to training the remaining parameters using a similar procedure. It should be noticed however that for the spatial size of the first CNN's output and the layer of the second CNN to match, we must feed to the first CNN a much larger input. Thus, training of the second CNN must be performed on larger patches. For example in the InputCascadeCNN (Fig. 3a), the input size to the first model is of size 65×65 which results into an output of size 33×33. Only in this case the outputs of the first CNN can be concatenated with the input channels of the second CNN.

4. Implementation details

Our implementation is based on the Pylearn2 library (Goodfellow et al., 2013a). 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.

Since CNN's are able to learn useful features from scratch, we applied only minimal pre-processing. We employed the same pre-processing as Tustison et al., the winner of the 2013 BRATS challenge (Menze et al., 2014). The pre-processing follows three steps. First, the 1% highest and lowest intensities are removed. Then, we apply an N4ITK bias correction (Avants et al., 2009) to T1 and T1C modalities. The data is then normalized within each input channel by subtracting the channel's mean and dividing by the channel's standard deviation.

As for post-processing, a simple method based on connected components was implemented to remove flat blobs which might appear in the predictions due to bright corners of the brains close to the skull.

The hyper-parameters of the different architectures (kernel and max pooling size for each layer and the number of layers) can be seen in Fig. 3. Hyper-parameters were tuned using grid search and cross-validation on a validation set (see Bengio (2012)). The chosen hyper-parameters were the ones for which the model performed best on the validation set. For max pooling, we always use a stride of 1. This is to keep per-pixel accuracy during full image prediction. We observed in practice that max pooling in the global path does not improve accuracy. We also found that adding additional layers to the architectures or increasing the capacity of the model by adding additional feature maps to the convolutional blocks do not provide any meaningful performance improvement.

Biases are initialized to zero except for the soft-max layer for which we initialized them to the log of the label frequencies. The kernels are randomly initialized from $U(-0.005, 0.005)$. Training takes about 3 minutes per epoch for the TwoPathCNN model on an NVIDIA Titan black card.

At test time, we run our code on a GPU in order to exploit its computational speed. Moreover, the convolutional nature of the output layer allows us to further accelerate computations at test time. This is done by feeding as input a full image and not individual patches. Therefore, convolutions at all layers can be extended to obtain all label probabilities $p(Y_{ij}|\mathbf{X})$ for the entire image. With this implementation, we are able to produce a segmentation in 25 s per brain on the Titan black card with the TwoPathCNN model. This turns out to be 45 times faster than when we extracted a patch at each pixel and processed them individually for the entire brain.

Predictions for the MFCascadeCNN model, the LocalCascadeCNN model, and InputCascadeCNN model take on average 1.5 minutes, 1.7 min and 3 min respectively.

5. Experiments and results

The experiments were carried out on real patient data obtained from the 2013 brain tumor segmentation challenge (BRATS2013), as part of the MICCAI conference (Farahani et al., 2014). The BRATS2013 data-set is comprised of 3 sub-data-sets. The training data-set, which contains 30 patient subjects all with pixel-accurate ground truth (20 high grade and 10 low grade tumors); the test data-set which contains 10 (all high grade tumors) and the leader-board data-set which contains 25 patient subjects (21 high grade and 4 low grade tumors). There is no ground truth provided for the test and leader-board data-sets. All brains in the data-set have the same orientation. For each brain there exists four modalities, namely T1, T1C, T2 and Flair which are co-registered. The training brains come with ground-truth for which five segmentation labels are provided, namely *non-tumor, necrosis, edema, non-enhancing tumor* and *enhancing tumor*. Fig. 4 shows an example of the data as well as the ground truth. In total, the model iterates over about 2.2 million examples of tumorous patches (this consists of all the 4 sub-tumor classes) and goes through 3.2 million of the healthy patches. As mentioned before during the first phase training, the distribution of examples introduced to the model from all five classes is uniform.

Please note that we could not use the BRATS 2014 data-set due to problems with both the system performing the evaluation and the quality of the labeled data. For these reasons the old BRATS 2014 data-set has been removed from the official web-site and, at the time of submitting this manuscript, the BRATS web-site still

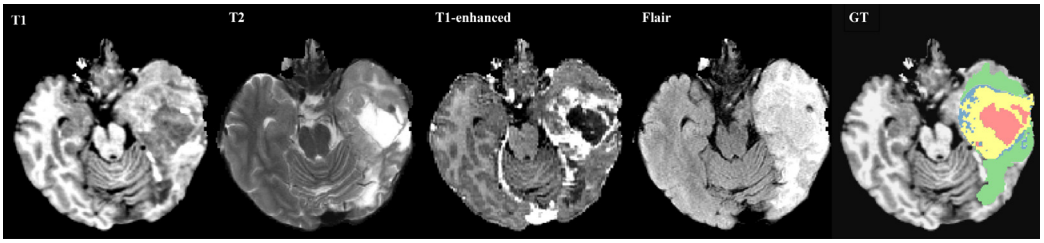


Fig. 4. The first four images from left to right show the MRI modalities used as input channels to various CNN models and the fifth image shows the ground truth labels where edema, enhanced tumor, necrosis, non-enhanced tumor.

showed: "Final data for BRATS 2014 to be released soon". Furthermore, we have even conducted an experiment where we trained our model with the old 2014 data-set and made predictions on the 2013 test data-set; however, the performance was worse than our results mentioned in this paper. For these reasons, we decided to focus on the BRATS 2013 data.

As mentioned in Section 3, we work with 2D slices due to the fact that the MRI volumes in the data-set do not posses an isotropic resolution and the spacing in the third dimension is not consistent across the data. We explored the use of 3D information (by treating the third dimension as extra input channels or by having an architecture which takes orthogonal slices from each view and makes the prediction on the intersecting center pixel), but that didn't improve performance and made our method very slow.

Note that as suggested by Krizhevsky et al. (2012), we applied data augmentation by flipping the input images. Unlike what was reported by Zeiler and Fergus (2014), it did not improve the overall accuracy of our model.

Quantitative evaluation of the models performance on the test set is achieved by uploading the segmentation results to the online BRATS evaluation system (Farahani et al., 2013). The online system provides the quantitative results as follows: The tumor structures are grouped in 3 different tumor regions. This is mainly due to practical clinical applications. As described by Menze et al. (2014), tumor regions are defined as:

(a) The *complete* tumor region (including all four tumor structures).
(b) The *core* tumor region (including all tumor structures exept "edema").
(c) The *enhancing* tumor region (including the "enhanced tumor" structure).

For each tumor region, *Dice* (identical to F measure), *Sensitivity* and *Specificity* are computed as follows:

$$Dice(P,T) = \frac{|P_1 \wedge T_1|}{(|P_1| + |T_1|)/2},$$
$$Sensitivity(P,T) = \frac{|P_1 \wedge T_1|}{|T_1|},$$
$$Specificity(P,T) = \frac{|P_0 \wedge T_0|}{|T_0|},$$

where P represents the model predictions and T represents the ground truth labels. We also note as T_1 and T_0 the subset of voxels predicted as positives and negatives for the tumor region in question. Similarly for P_1 and P_0. The online evaluation system also provides a ranking for every method submitted for evaluation. This includes methods from the 2013 BRATS challenge published in (Menze et al., 2014) as well as anonymized unpublished methods for which no reference is available. In this section, we report experimental results for our different CNN architectures.

5.1. The TwoPathCNN architecture

As mentioned previously, unlike conventional CNNs, the TwoPathCNN architecture has two pathways: a "local" path focusing on details and a "global" path more focused on the context. To better understand how joint training of the global and local pathways benefits the performance, we report results on each pathway as well as results on averaging the outputs of each pathway when trained separately. Our method also deals with the unbalanced nature of the problem by training in two phases as discussed in Section 3.2.0.2. To see the impact of the two phase training, we report results with and without it. We refer to the CNN model consisting of only the local path (i.e. conventional CNN architecture) as LocalPathCNN, the CNN model consisting of only the global path as GlobalPathCNN, the model averaging the outputs of the local and global paths (i.e. LocalPathCNN and GlobalPathCNN) as AverageCNN and the two-pathway CNN architecture as TwoPathCNN. The second training phase is noted by appending '*' to the architecture name. Since the second phase training has a substantial effect and always improves the performance, we only report results on GlobalPathCNN and AverageCNN with the second phase.

Table 1 presents the quantitative results of these variations. This table contains results for the TwoPathCNN with one and two training phases, the common single path CNN (i.e. LocalPathCNN) with one and two training phases, the GlobalPathCNN* which is a single path CNN model following the global pathway architecture and the output average of each of the trained single-pathway models (AverageCNN*). Without much surprise, the single path with one training phase CNN was ranked last with the lowest scores on almost every region. Using a second training phase gave a significant boost to that model with a rank that went from 15 to 9. Also, the table shows that joint training of the local and global paths yields better performance compared to when each pathway is trained separately and the outputs are averaged. One likely explanation is that by joint training the local and global paths, the model allows the two pathways to co-adapt. In fact, the AverageCNN* performs worse than the LocalPathCNN* due to the fact that the GlobalPathCNN* performs very badly. The top performing method in the uncascaded models is the TwoPathCNN* with a rank of 4.

Also, in some cases results are less accurate over the Enhancing region than for the Core and Complete regions. There are two main reasons for that. First, borders are usually diffused and there are no clear cut between enhanced tumor and non-enhanced tissues. This creates problems for both user labeling, ground truth, as well as the model. The second reason is that the model learns what it sees in the ground truth. Since the labels are created by different people and since the borders are not clear, each user has a slightly different interpretation of the borders of the enhanced tumor and so sometimes we see overly thick enhanced tumor in the ground truth.

Fig. 5 shows representation of low level features in both local and global paths. As seen from this figure, features in the local

Table 1
Performance of the TwoPathCNN model and variations. The second phase training is noted by appending '*' to the architecture name. The 'Rank' column represents the ranking of each method in the online score board at the time of submission.

Rank	Method	Dice			Specificity			Sensitivity		
		Complete	Core	Enhancing	Complete	Core	Enhancing	Complete	Core	Enhancing
4	TwoPathCNN*	0.85	0.78	0.73	0.93	0.80	0.72	0.80	0.76	0.75
9	LocalPathCNN*	0.85	0.74	0.71	0.91	0.75	0.71	0.80	0.77	0.73
10	AverageCNN*	0.84	0.75	0.70	0.95	0.83	0.73	0.77	0.74	0.73
14	GlobalPathCNN*	0.82	0.73	0.68	0.93	0.81	0.70	0.75	0.65	0.70
14	TwoPathCNN	0.78	0.63	0.68	0.67	0.50	0.59	0.96	0.89	0.82
15	LocalPathCNN	0.77	0.64	0.68	0.65	0.52	0.60	0.96	0.87	0.80

Table 2
Performance of the cascaded architectures. The reported results are from the second phase training. The 'Rank' column shows the ranking of each method in the online score board at the time of submission.

Rank	Method	Dice			Specificity			Sensitivity		
		Complete	Core	Enhancing	Complete	Core	Enhancing	Complete	Core	Enhancing
2	InputCascadeCNN*	0.88	0.79	0.73	0.89	0.79	0.68	0.87	0.79	0.80
4-a	MFCascadeCNN*	0.86	0.77	0.73	0.92	0.80	0.71	0.81	0.76	0.76
4-c	LocalCascadeCNN*	0.88	0.76	0.72	0.91	0.76	0.70	0.84	0.80	0.75

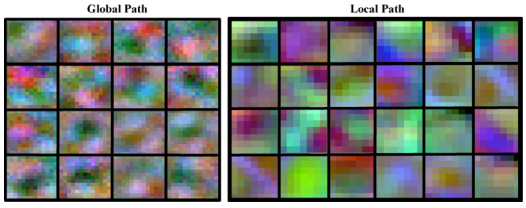


Fig. 5. Randomly selected filters from the first layer of the model. From left to right the figure shows visualization of features from the first layer of the global and local path respectively. Features in the local path include more edge detectors while the global path contains more localized features.

path include more edge detectors while the ones in the global path are more localized features. Unfortunately, visualizing the learned mid/high level features of a CNN is still very much an open research problem. However, we can study the impact these features have on predictions by visualizing the segmentation results of different models. The segmentation results on two subjects from our validation set, produced by different variations of the basic model can be viewed in Fig. 7.[4] As shown in the figure, the two-phase training procedure allows the model to learn from a more realistic distribution of labels and thus removes false positives produced by the model which trains with one training phase. Moreover, by having two pathways, the model can simultaneously learn the global contextual features as well as the local detailed features. This gives the advantage of correcting labels at a global scale as well as recognizing fine details of the tumor at a local scale, yielding a better segmentation as oppose to a single path architecture which results in smoother boundaries. Joint training of the two convolutional pathways and having two training phases achieves better results.

5.2. Cascaded architectures

We now discuss our experiments with the three cascaded architectures namely InputCascadeCNN, LocalCascadeCNN and MFCascadeCNN. Table 2 provides the quantitative results for each architecture. Fig. 7 also provides visual examples of the segmentation generated by each architecture.

We find that the MFCascadeCNN* model yields smoother boundaries between classes. We hypothesize that, since the neurons in the soft-max output layer are directly connected to the previous outputs within each receptive field, these parameters are more likely to learn that the center pixel label should have a similar label to its surroundings.

As for the LocalCascadeCNN* architecture, while it resulted in fewer false positives in the complete tumor category, the performance in other categories (i.e. tumor core and enhanced tumor) did not improve.

Fig. 8 shows segmentation results from the same brains (as in Fig. 7) in Sagittal and Coronal views. The InputCascadeCNN* model was used to produce these results. As seen from this figure, although the segmentation is performed on Axial view but the output is consistent in Coronal and Sagittal views. Although subjects in Figs. 5 and 6 are from our validation set for which the model is not trained on and the segmentation results from these subjects can give a good estimate of the models performance on a test set, however, for further clarity we visualise the models performance on two subjects from BRATS-2013 testset. These results are shown in Fig. 9 in Saggital (top) and Axial (bottom) views.

To better understand the process for which InputCascadeCNN* learns features, we present in Fig. 6 the progression of the model by making predictions at every few epochs on a subject from our validation set.

Overall, the best performance is reached by the InputCascadeCNN* model. It improves the Dice measure on all tumor regions. With this architecture, we were able to reach the second rank on the BRATS 2013 scoreboard. While MFCascadeCNN*, TwoPathCNN* and LocalCascadeCNN* are all ranked 4, the inner ranking between these three models is noted as 4a, 4b and 4c respectively.

Table 3 shows how our implemented architectures compare with currently published state-of-the-art methods as mentioned in (Menze et al., 2014).[5] The table shows that InputCascadeCNN* out performs Tustison et al. the winner of the BRATS 2013 challenge and is ranked first in the table. Results from the BRATS-2013 leaderboard presented in Table 4 shows that our method outper-

[4] It is important to note that we do not train the model on the validation set and thus the quality of the results is not due to overfitting

[5] Please note that the results mentioned in Tables 3 and 4 and are from methods competing in the BRATS 2013 challenge for which a static table is provided [https://www.virtualskeleton.ch/BRATS/StaticResults2013]. Since then, other methods have been added to the score board but for which no reference is available.

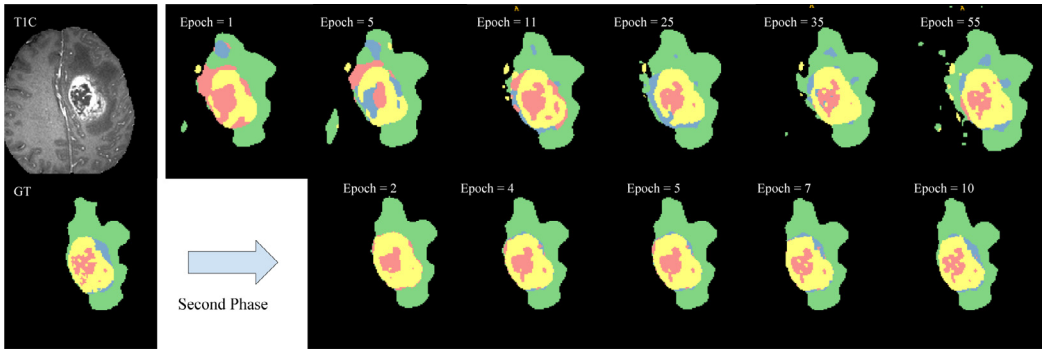


Fig. 6. Progression of learning in InputCascadeCNN*. The stream of figures on the first row from left to right show the learning process during the first phase. As the model learns better features, it can better distinguish boundaries between tumor sub-classes. This is made possible due to uniform label distribution of patches during the first phase training which makes the model believe all classes are equiprobable and causes some false positives. This drawback is alleviated by training a second phase (shown in second row from left to right) on a distribution closer to the true distribution of labels. The color codes are as follows: edema, enhanced tumor, necrosis, non-enhanced tumor.

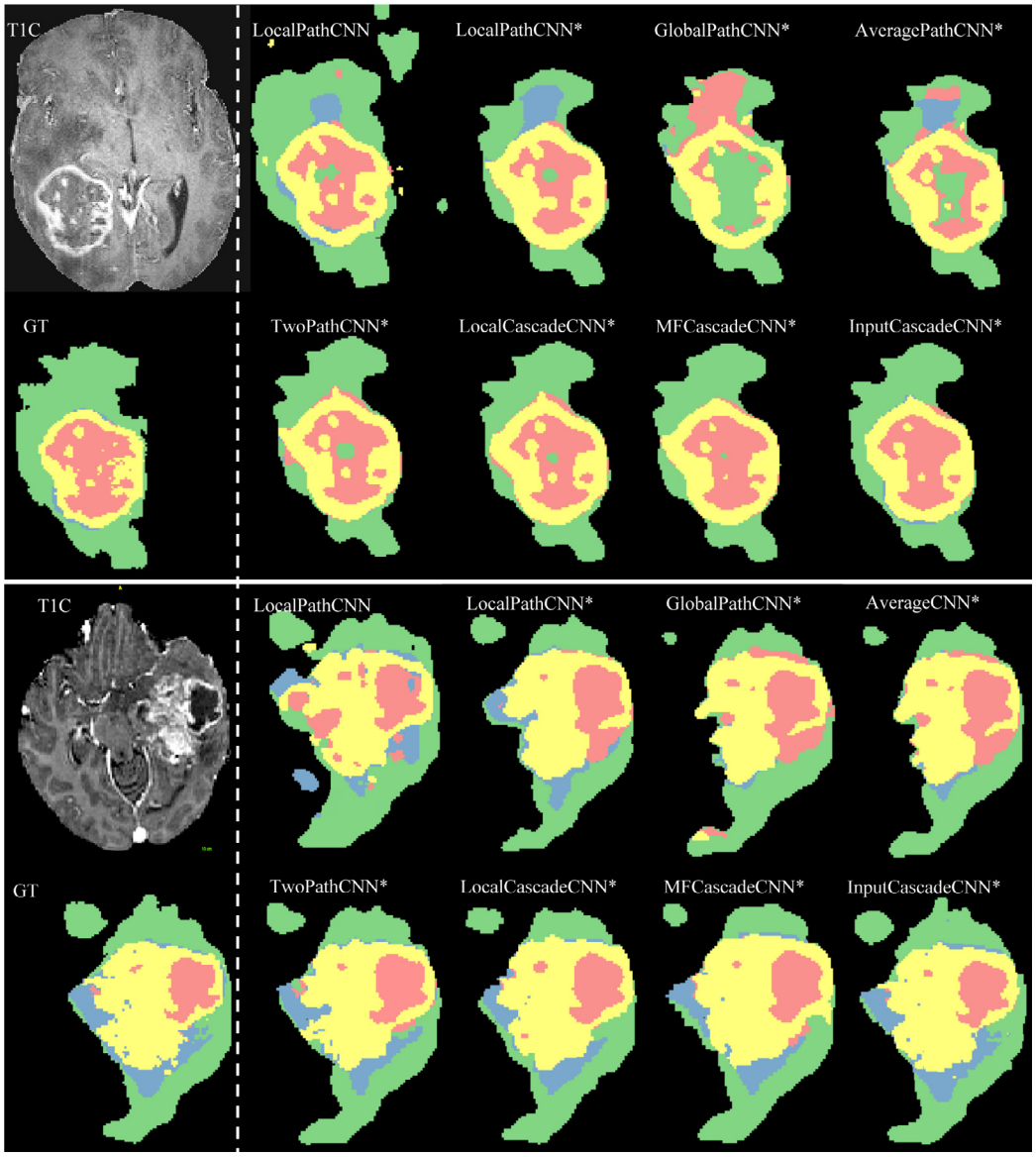


Fig. 7. Visual results from our CNN architectures from the Axial view. For each sub-figure, the top row from left to right shows T1C modality, the conventional one path CNN, the Conventional CNN with two training phases, and the TwoPathCNN model. The second row from left to right shows the ground truth, LocalCascadeCNN model, the MFCascadeCNN model and the InputCascadeCNN. The color codes are as follows: edema, enhanced tumor, necrosis, non-enhanced tumor.

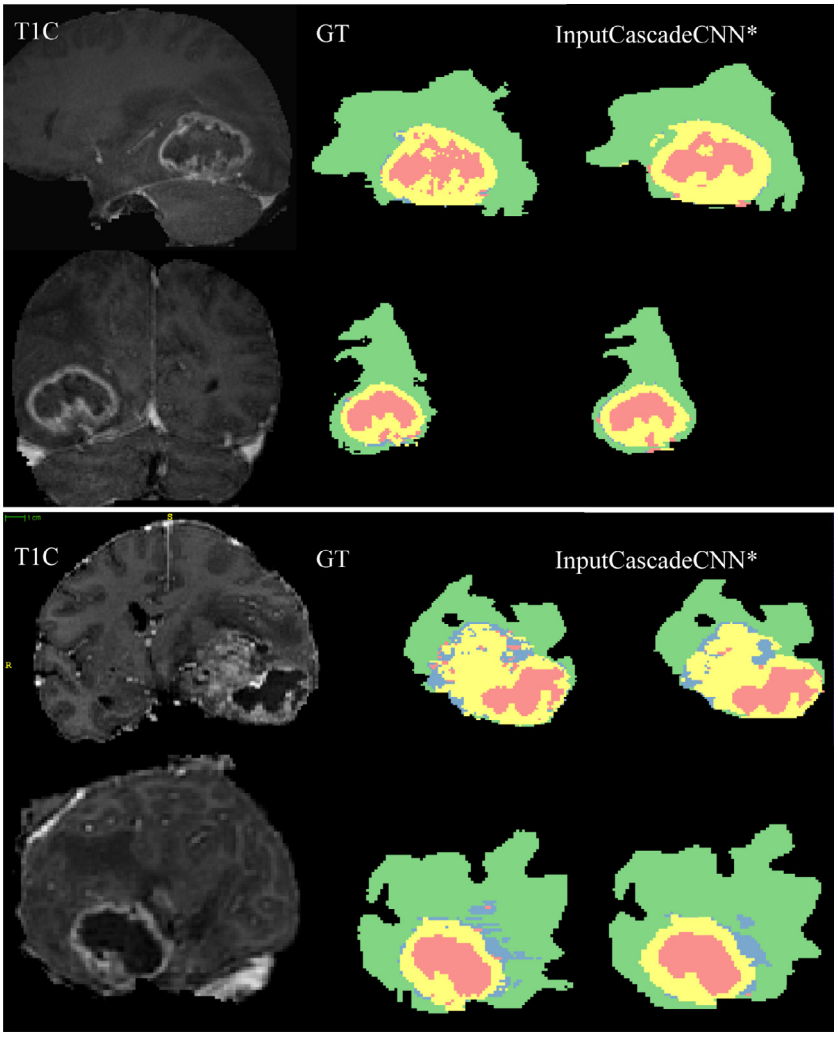


Fig. 8. Visual results from our top performing model, InputCascadeCNN* on Coronal and Sagittal views. The subjects are the same as in Fig. 7. In every sub-figure, the top row represents the Sagital view and the bottom row represents the Coronal view. The color codes are as follows: edema, enhanced tumor, necrosis, non-enhanced tumor.

Table 3
Comparison of our implemented architectures with the state-of-the-art methods on the BRATS-2013 test set.

Method	Dice			Specificity			Sensitivity		
	Complete	Core	Enhancing	Complete	Core	Enhancing	Complete	Core	Enhancing
InputCascadeCNN*	0.88	0.79	0.73	0.89	0.79	0.68	0.87	0.79	0.80
Tustison	0.87	0.78	0.74	0.85	0.74	0.69	0.89	0.88	0.83
MFCascadeCNN*	0.86	0.77	0.73	0.92	0.80	0.71	0.81	0.76	0.76
TwoPathCNN*	0.85	0.78	0.73	0.93	0.80	0.72	0.80	0.76	0.75
LocalCascadeCNN*	0.88	0.76	0.72	0.91	0.76	0.70	0.84	0.80	0.75
LocalPathCNN*	0.85	0.74	0.71	0.91	0.75	0.71	0.80	0.77	0.73
Meier	0.82	0.73	0.69	0.76	0.78	0.71	0.92	0.72	0.73
Reza	0.83	0.72	0.72	0.82	0.81	0.70	0.86	0.69	0.76
Zhao	0.84	0.70	0.65	0.80	0.67	0.65	0.89	0.79	0.70
Cordier	0.84	0.68	0.65	0.88	0.63	0.68	0.81	0.82	0.66
TwoPathCNN	0.78	0.63	0.68	0.67	0.50	0.59	0.96	0.89	0.82
LocalPathCNN	0.77	0.64	0.68	0.65	0.52	0.60	0.96	0.87	0.80
Festa	0.72	0.66	0.67	0.77	0.77	0.70	0.72	0.60	0.70
Doyle	0.71	0.46	0.52	0.66	0.38	0.58	0.87	0.70	0.55

forms other approaches on this dataset. We also compare our top performing method in Table 5 with state-of-the-art methods on BRATS-2012, "4 label" test set as mentioned in (Menze et al., 2014). As seen from this table, our method out performs other methods in the tumor Core category and gets competitive results on other categories.

Let us mention that Tustison's method takes 100 min to compute predictions per brain as reported in Menze et al. (2014), while the InputCascadeCNN* takes 3 min, thanks to the fully convolutional architecture and the GPU implementation, which is over 30 times faster than the winner of the challenge. The TwoPathCNN* has a performance close to the state-of-the-art. However, with a

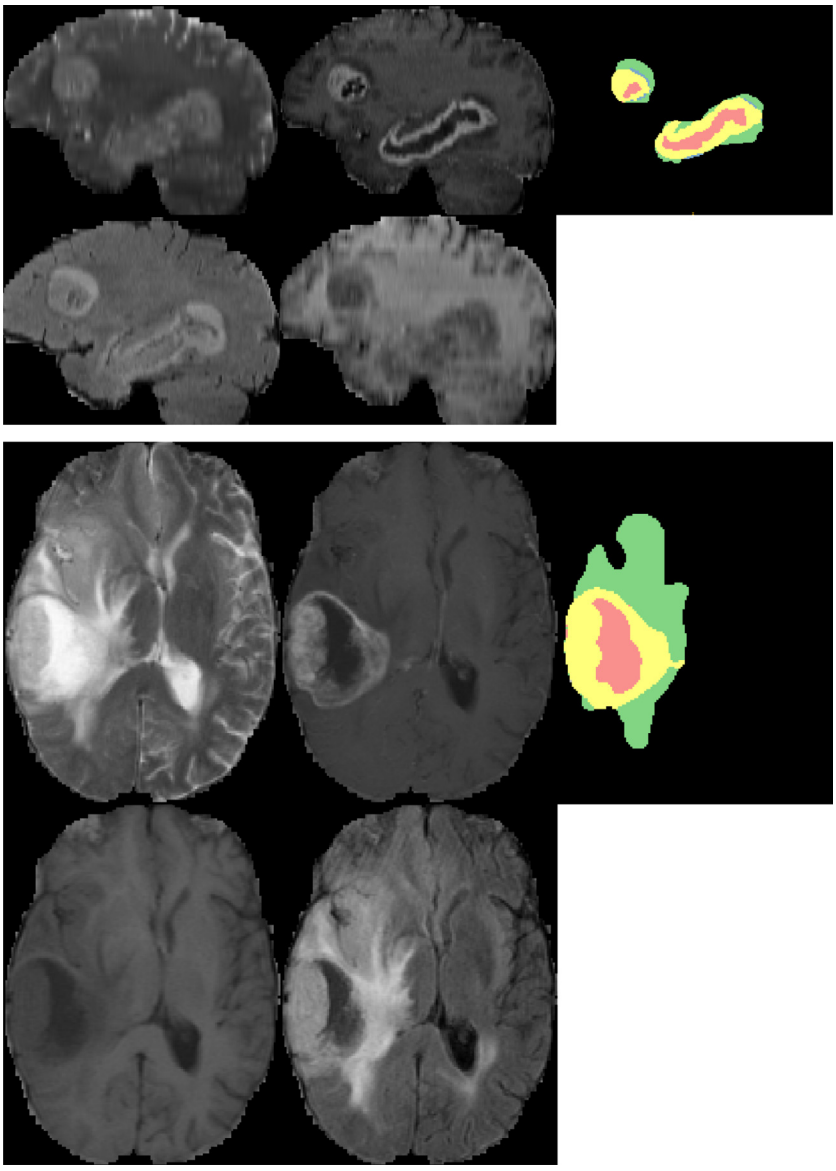


Fig. 9. Visual segmentation results from our top performing model, InputCascadeCNN*, on examples of the BRATS2013 test data-set in Saggital (top) and Axial (bottom) views. The color codes are as follows: edema, enhanced tumor, necrosis, non-enhanced tumor.

Table 4
Comparison of our top implemented architectures with the state-of-the-art methods on the BRATS-2013 leaderboard set.

Method	Dice			Specificity			Sensitivity		
	Complete	Core	Enhancing	Complete	Core	Enhancing	Complete	Core	Enhancing
InputCascadeCNN*	0.84	0.71	0.57	0.88	0.79	0.54	0.84	0.72	0.68
Tustison	0.79	0.65	0.53	0.83	0.70	0.51	0.81	0.73	0.66
Zhao	0.79	0.59	0.47	0.77	0.55	0.50	0.85	0.77	0.53
Meier	0.72	0.60	0.53	0.65	0.62	0.48	0.88	0.69	0.6
Reza	0.73	0.56	0.51	0.68	0.64	0.48	0.79	0.57	0.63
Cordier	0.75	0.61	0.46	0.79	0.61	0.43	0.78	0.72	0.52

prediction time of 25 s, it is over 200 times faster than Tustison's method. Other top methods in the table are that of Meier et al and Reza et al with processing times of 6 and 90 min respectively. Recently Subbanna et al. (2014), published competitive results on the BRATS 2013 dataset, reporting dice measures of 0.86, 0.86, 0.77 for Complete, Core and Enhancing tumor regions. Since they do not report Specificity and Sensitivity measures, a completely fair comparison with that method is not possible. However, as mentioned in Subbanna et al. (2014), their method takes 70 min to process a subject, which is about 23 times slower than our method.

Regarding other methods using CNNs, Urban et al. (2014) used an average of two 3D convolutional networks with dice measures of 0.87, 0.77, 0.73 for Complete, Core and Enhancing tumor regions on BRATS 2013 test dataset with a prediction time of about 1 minute per model which makes for a total of 2 min. Again, since they do not report Specificity and Sensitivity measures, we can not make a full comparison. However, based on their dice scores our TwoPathCNN* is similar in performance while taking only 25 s, which is four times faster. And the InputCascadeCNN* is better or equal in accuracy while having the same processing time. As

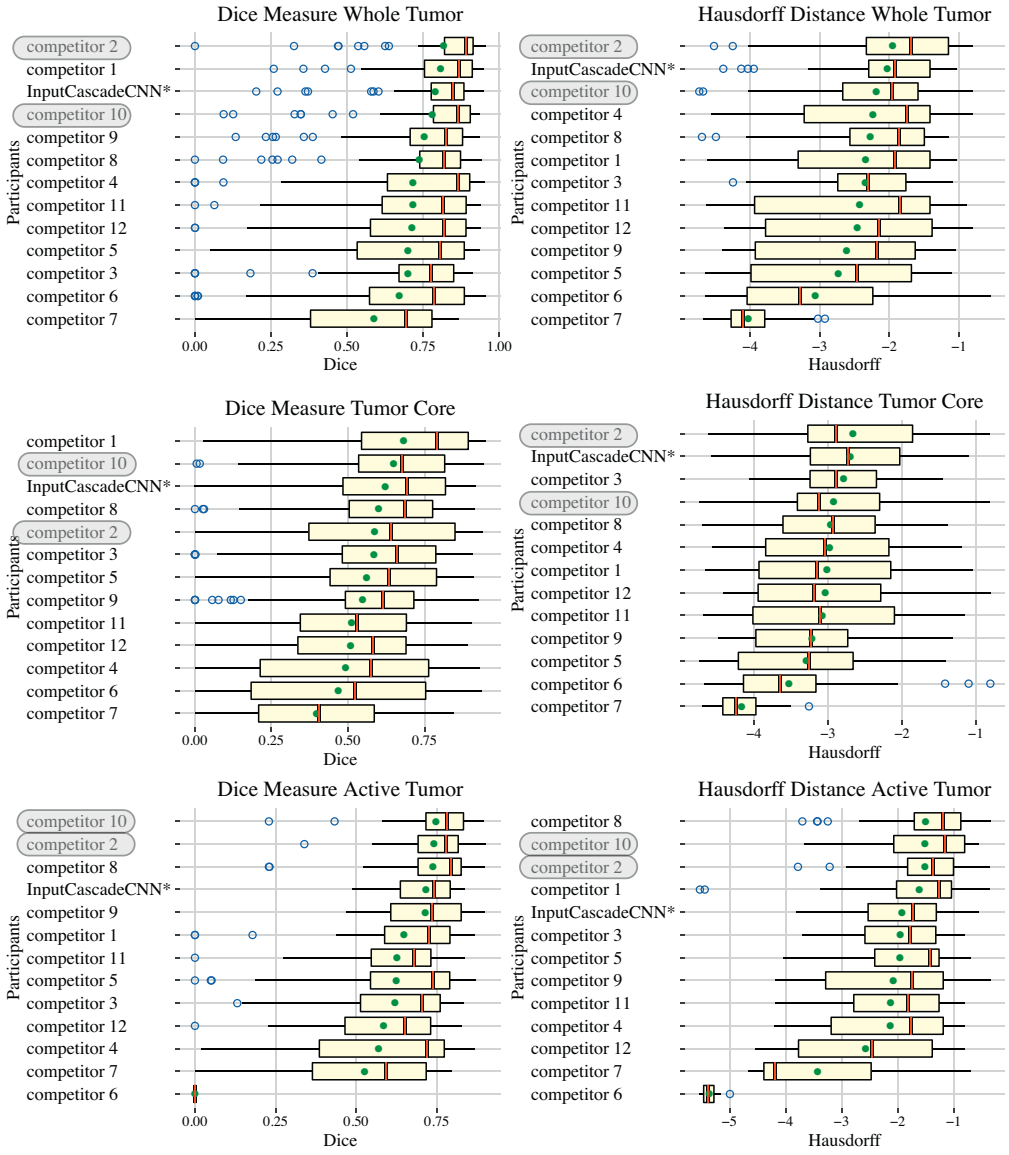


Fig. 10. Our BRATS'15 challenge results using InputCascadeCNN*. Dice scores and negative log Hausdorff distances are presented for the three tumor categories. Since the results of the challenge are not yet publicly available, we are unable to disclose the name of the participants. The semi-automatic methods are highlighted in gray. In each sub-figure, the methods are ranked based on the mean value. The mean is presented in green, the median in red and outliers in blue. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 5
Comparison of our top implemented architectures with the state-of-the-art methods on the BRATS-2012 "4 label" test set as discussed in Menze et al. (2014).

Method	Dice		
	Complete	Core	Enhancing
InputCascadeCNN*	0.81	0.72	0.58
Subbanna	0.75	0.70	0.59
Zhao	0.82	0.66	0.42
Tustison	0.75	0.55	0.52
Festa	0.62	0.50	0.61

for (Zikic et al., 2014), they do not report results on BRATS 2013 test data-set. However, their method is very similar to the LocalPathCNN which, according to our experiments, has worse performance.

Using our best performing method, we took part in the BRATS 2015 challenge. The BRATS 2015 training data-set comprises of 220 subjects with high grade and 54 subjects with low grade gliomas. There are 53 subjects with mixed high and low grade gliomas for testing. Every participating group had 48 h- from receiving the test subjects to process them and submit their segmentation results to the online evaluation system. BRATS'15 contains the training data of 2013. The ground truth for the rest of the training brains is generated by a voted average of segmented results of the top performing methods in BRATS'13 and BRATS'12. Some of these automatically generated ground truths have been refined manually by a user.

Because distribution of the intensity values in this dataset is very variable from one subject to another, we used a 7 fold cross validation for training. At test time, a voted average of these models was made to make prediction for each subject in the test dataset. The results of the challenge are presented in Fig. 10. The semi-automatic methods participating in the challenge have been

highlighted in grey. Please note since these results are not yet publicly available, we refrain from disclosing the name of the participants. In this figure the semi-automatic methods are highlighted in gray. As seen from the figure, our method ranks either first or second on Complete tumor and tumor Core categories and gets competitive results on active tumor category. Our method has also less outliers than most other approaches.

6. Conclusion

In this paper, we presented an automatic brain tumor segmentation method based on deep convolutional neural networks. We considered different architectures and investigated their impact on the performance. Results from the BRATS 2013 online evaluation system confirms that with our best model we managed to improve on the currently published state-of-the-art method both on accuracy and speed as presented in MICCAI 2013. The high performance is achieved with the help of a novel two-pathway architecture (which can model both the local details and global context) as well as modeling local label dependencies by stacking two CNN's. Training is based on a two phase procedure, which we've found allows us to train CNNs efficiently when the distribution of labels is unbalanced.

Thanks to the convolutional nature of the models and by using an efficient GPU implementation, the resulting segmentation system is very fast. The time needed to segment an entire brain with any of the these CNN architectures varies between 25 seconds and 3 minutes, making them practical segmentation methods.

References

Alvarez, J.M., Gevers, T., LeCun, Y., Lopez, A.M., 2012. Road scene segmentation from a single image. In: Proceedings of the 12th European Conference on Computer Vision - Volume Part VII. Springer-Verlag, Berlin, Heidelberg, pp. 376–389.

Angelini, E., Clatz, O.E., Konukoglu, E., Capelle, L., Duffau, H., 2007. Glioma dynamics and computational models: A review of segmentation, registration, and in silico growth algorithms and their clinical applications. Curr. Med. Imaging Rev. 3 (4), 262–276.

Avants, B.B., Tustison, N., Song, G., 2009. Advanced normalization tools (ants). Insight J.

Bauer, S., Nolte, L.-P., Reyes, M., 2011. Fully automatic segmentation of brain tumor images using support vector machine classification in combination with hierarchical conditional random field regularization.. In: MICCAI, Vol. 6893, pp. 354–361.

Bauer, S., Wiest, R., Nolte, L., Reyes, M., 2013. A survey of mri-based medical image analysis for brain tumor studies. Phys. Med. Biol. 58, 97–129.

Bengio, Y., 2012. Practical recommendations for gradient-based training of deep architectures. In: Neural Networks: Tricks of the Trade. Springer, pp. 437–478.

Bengio, Y., Courville, A., Vincent, P., 2013. Representation learning: a review and new perspectives. Pattern Anal. Mach. Intell. IEEE Trans. 35, 1798–1828.

Ciresan, D., Giusti, A., Gambardella, L.M., Schmidhuber, J., 2012. Deep neural networks segment neuronal membranes in electron microscopy images. In: Advances in Neural Information Processing Systems, pp. 2843–2851.

Clark, M., Hall, L., Goldgof, D., Velthuizen, R.P., Murtagh, F., Silbiger, M.L., 1998. Automatic tumor segmentation using knowledge-based clustering. IEEE Trans. Med. Imag. 17, 187–201.

Cobzas, D., Birkbeck, N., Schmidt, M., Jgersand, M., Murtha, A., 2007. 3D variational brain tumor segmentation using a high dimensional feature set. In: ICCV, pp. 1–8.

Davy, A., Havaei, M., Warde-Farley, D., Biard, A., Tran, L., Jodoin, P.-M., Courville, A., Larochelle, H., Pal, C., Bengio, Y., 2014. Brain tumor segmentation with deep neural networks. Proc. BRATS-MICCAI.

Doyle, S., Vasseur, F., Dojat, M., Forbes, F., 2013. Fully automatic brain tumor segmentation from multiple mr sequences using hidden markov fields and variational em. Proc. BRATS-MICCAI.

Farabet, C., Couprie, C., Najman, L., LeCun, Y., 2013. Learning hierarchical features for scene labeling. Pattern Anal. Mach. Intell. IEEE Trans. 35, 1915–1929.

Farahani, K., Menze, B., Reyes, M., 2013. Multimodal Brain Tumor Segmentation (BRATS 2013).

Farahani, K., Menze, B., Reyes, M., 2014. Brats 2014 Challenge Manuscripts.

Glorot, X., Bordes, A., Bengio, Y., 2011. Domain adaptation for large-scale sentiment classification: a deep learning approach. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 513–520.

Goodfellow, I.J., Warde-Farley, D., Lamblin, P., Dumoulin, V., Mirza, M., Pascanu, R., Bergstra, J., Bastien, F., Bengio, Y., 2013. Pylearn2: a machine learning research library. arXiv preprint arXiv:1308.4214.

Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y., 2013. Max-out networks. ICML.

Gotz, M., Weber, C., Blocher, J., Stieltjes, B., Meinzer, H.-P., Maier-Hein, K., 2014. Extremely randomized trees based brain tumor segmentation. In: in Proceedings of BRATS Challenge - MICCAI.

Hamamci, A., Kucuk, N., Karaman, K., Engin, K., Unal, G., 2012. Tumor-cut: Segmentation of brain tumors on contrast enhanced mr images for radiosurgery applications. IEEE Trans. Med. Imag. 31, 790–804.

Hariharan, B., Arbeláez, P., Girshick, R., Malik, J., 2014. Simultaneous detection and segmentation. In: Computer Vision–ECCV 2014. Springer, pp. 297–312.

Havaei, M., Jodoin, P.-M., Larochelle, H., 2014. Efficient interactive brain tumor segmentation as within-brain knn classification. In: International Conference on Pattern Recognition (ICPR).

Huang, G.B., Jain, V., 2013. Deep and wide multiscale recursive networks for robust image labeling. arXiv preprint arXiv:1310.0354.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., LeCun, Y., 2009. What is the best multi-stage architecture for object recognition? In: Computer Vision, 2009 IEEE 12th International Conference on. IEEE, pp. 2146–2153.

Khotanlou, H., Colliot, O., Atif, J., Bloch, I., 2009. 3D brain tumor segmentation in mri using fuzzy classification, symmetry analysis and spatially constrained deformable models. Fuzzy Sets Syst. 160, 1457–1473.

Kleesiek, J., Biller, A., Urban, G., Kothe, U., Bendszus, M., Hamprecht, F.A., 2014. ilastik for multi-modal brain tumor segmentation. Proc. BRATS-MICCAI.

Krizhevsky, A., Sutskever, I., Hinton, G., 2012. ImageNet classification with deep convolutional neural networks. NIPS.

Kwon, D., Akbari, H., Da, X., Gaonkar, B., Davatzikos, C., 2014. Multi-modal brain tumor image segmentation using glistr. In: in Proceedings of BRATS Challenge - MICCAI.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86, 2278–2324.

Lee, C.-H., Schmidt, M., Murtha, A., Bistritz, A., S, J., Greiner, R., 2005. Segmenting brain tumor with conditional random fields and support vector machines. In: in Proceedings of Workshop on Computer Vision for Biomedical Image Applications.

Long, J., Shelhamer, E., Darrell, T., 2015. Fully convolutional networks for semantic segmentation. CVPR (to appear).

Menze, B., Reyes, M., Leemput, K.V., 2014. The multimodal brain tumor image segmentation benchmark (brats). IEEE Trans. Med. Imag..

Tustison, N., Wintermark, M.C.D., Avants, B., 2013. Ants and árboles. In: in Proceedings of BRATS Challenge - MICCAI.

Parisot, S., Duffau, H., Chemouny, S., Paragios, N., 2012. Joint tumor segmentation and dense deformable registration of brain mr images.. In: MICCAI, Vol. 7511, pp. 651–658.

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.

Popuri, K., Cobzas, D., Murtha, A., Jgersand, M., 2012. 3D variational brain tumor segmentation using dirichlet priors on a clustered feature set.. Int. J. Comput. Assist. Radiol. Surg. 7, 493–506.

Prastawa, M., Bullit, E., Ho, S., Gerig, G., 2004. A brain tumor segmentation framework based on outlier detection. Med. Image Anal. 8, 275–283.

Prastawa, M., Bullitt, E., Ho, S., Gerig, G., 2003. Robust estimation for brain tumor segmentation. In: Medical Image Computing and Computer-Assisted Intervention-MICCAI 2003. Springer, pp. 530–537.

Meier, R., Bauer, S., Slotboom, J., Wiest, R., Reyes, M., 2014. Appearance- and context-sensitive features for brain tumor segmentation. in proc of BRATS Challenge - MICCAI.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1988. Learning representations by back-propagating errors. Cognit. Mode. 5.

Schmidt, M., Levner, I., Greiner, R., Murtha, A., Bistritz, A., 2005. Segmenting brain tumors using alignment-based features. In: Int. Conf on Machine Learning and Applications, pp. 6–pp.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15, 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

Subbanna, N., Precup, D., Arbel, T., 2014. Iterative multilevel mrf leveraging context and voxel information for brain tumour segmentation in mri.

Subbanna, N., Precup, D., Collins, L., Arbel, T., 2013. Hierarchical probabilistic gabor and mrf segmentation of brain tumours in mri volumes.. In: in Proceedings of MICCAI, Vol. 8149, pp. 751–758.

Urban, G., Bendszus, M., Hamprecht, F., Kleesiek, J., 2014. Multi-modal brain tumor segmentation using deep convolutional neural networks. Proc. BRATS-MICCAI.

Xing, E.P., Jordan, M.I., Russell, S., 2002. A generalized mean field algorithm for variational inference in exponential families. In: Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann Publishers Inc., pp. 583–591.

Zeiler, M.D., Fergus, R., 2014. Visualizing and understanding convolutional networks. In: Computer Vision–ECCV 2014. Springer, pp. 818–833.

Zikic, D., Glocker, B., Konukoglu, E., Criminisi, A., Demiralp, C., Shotton, J., Thomas, O., Das, T., Jena, R., Price, S., 2012. Decision forests for tissue-specific segmentation of high-grade gliomas in multi-channel mr. In: Medical Image Computing and Computer-Assisted Intervention–MICCAI 2012. Springer, pp. 369–376.

Zikic, D., Ioannou, Y., Brown, M., Criminisi, A., 2014. Segmentation of brain tumor tissues with convolutional neural networks. Proc. BRATS-MICCAI.