Brain tumor segmentation with Deep Neural Networks

Bar Ettedgui **308026459** **barettedgui@mail.tau.ac.il**
Yuval Buchsweiler **300079985** **buchsweiler@mail.tau.ac.il**

A b s t r a c t
Automated segmentation of brain tumors from 3D magnetic resonance images (MRIs) is necessary for the diagnosis, monitoring, and treatment planning of the disease. Glioblastomas (type of brain tumor), which our network is tailored for, can appear anywhere in the brain, in different sizes, shapes and contrasts. Manual labeling practices require anatomical knowledge, are expensive, time consuming and can be inaccurate due to human error. For these reasons, coming up with a machine learning solution for this problem that is both accurate, fast and efficient is imperative. In this paper, we present a cascaded brain tumor segmentation architecture in which the output of a basic CNN model is treated as an additional source of information for a subsequent CNN model of the same type. Training is conducted over 2D randomly selected patches, while we use complete 2D slices for fast inference. Our work is based on the article "Brain tumor segmentation with Deep Neural Networks" (Havaei et al., 2017) [1]. We implemented our own version of the architecture which achieved the best results in the original paper and went on to test some "deeper" architectures to try and improve the existing results. The architectures were trained on the BRaTS (Brain Tumor Segmentation) 2013 dataset and our best results were on par with those presented in the original paper while inference with our architecture is approximately about twice as fast, taking less than 15 seconds per brain.

1. Introduction

Each year, more and more people are diagnosed with brain cancer. It is estimated that approximately 24,000 new cases of brain cancer will be diagnosed in the United States alone. Brain tumors are categorized into primary and secondary tumor types. Primary brain tumors originate from brain cells, whereas secondary tumors metastasize into the brain from other organs. The most common type of primary brain tumors are gliomas, which arise from brain glial cells. Gliomas can be of low-grade (LGG) and high-grade (HGG) subtypes. High grade gliomas are an aggressive type of malignant brain tumor that grow rapidly, usually require surgery and radiotherapy and have poor survival prognosis.

The great numbers of people suffering from brain tumors requires an automatic solution that will at least aid the work of radiologists in doing the labeling over the vast amounts of data. Automated segmentation of 3D brain tumors can save physicians time and provide an accurate reproducible solution for further tumor analysis and monitoring. MRI provides images with great details of the brain and is commonly used for brain tumor diagnosis. All the more, brain tumor segmentation from MR images can have great impact for improved diagnostics, growth rate prediction and treatment planning.

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

Gliomas and Glioblastomas are much more difficult to localize compared to other types of brain cancers like meningiomas. These tumors are often diffused, poorly contrasted, and extend tentacle like structures that make them difficult to segment. They can appear anywhere in the brain, in almost any shape and size. Another challenge is the non-standardized voxel values in MR images, which may have great variability between different images acquired using different MR machines and different practices. Hospitals use different MR equipment and acquisition protocols and as a result, voxel values are not restricted to the range of 0-255 as in grayscale images and may have values in the thousands and even millions for certain outliers.

Since glioblastomas are infiltrative tumors, their borders are often fuzzy and hard to distinguish from healthy tissues. In order to be able to distinguish between healthy tissue and the 3 type of tumorous tissue, 4 modalities are used, each represented as a separate image channel: T1 (spin-lattice relaxation), T1-contrast enhanced (T1C), T2 (spin-spin relaxation) and fluid attenuation inversion recovery (FLAIR) pulse sequences. The contrast between the different modalities provides a signature which is almost unique to each tissue type. For example, active tumor regions will be brighter and necrotic tissue will be darker compared to its surrounding tissues while using the T1 modality, while edema tissue will be brighter while using the T2 modality.

Another problem in brain tumor segmentation is the great imbalance within the dataset between the different classes. Approximately 99% of the voxels are of class 0 (healthy tissue or background) while the remaining 1% are comprised of the remaining 3 tumor classes. We had to take this imbalance into account, and we chose to do it in a different way than the one used in the original paper (more information under the "Training" section).

2. Related Work

The BraTS challenge is a well-known challenge that has been running yearly since 2012. Each year, different groups submit different architectures in an effort to provide the best segmentation results on the BraTS dataset.

At the time the original article was released, brain tumor segmentation using deep neural networks was in its infancy. Most previous works used discriminative models and classical methods to extract low level features from a given brain image and model the relationship between these features and the label of a given voxel. These methods were in most cases based on hand-crafted features and required the computation of a large number of features in order to be accurate when used with many traditional machine learning techniques. This made them slow to compute and expensive memory-wise. At the time, in the BraTS 2014 challenge, deep CNN's started to be explored in the context of brain tumor segmentations and showed promising results. Multiple teams have used methods based on convolutional neural networks - Davy et al. (2014) [1]; Zikic et al. (2014) [2]; Urban et al. (2014) [3]. All three methods divided 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 the center pixel class of each patch. 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.

It can be seen that the architectures at the time were relatively simple and used small patches and mainly 2D images which were extracted from the 3D MR volumes. This was mainly because of the limited resources and the computational limits of GPU's from that period.

Since the original article was published, there have been great advancements in the field of deep learning and architectures have become much more advanced in comparison to the relatively simple and shallow architecture proposed in the article. In the 2017 BraTS challenge, the top performing submissions included Kamnitsas et al. (2017) [4] who proposed to ensemble several models for robust segmentation (EMMA), and Wang et al. (2017) [5] who proposed to segment tumor subregions in cascade using anisotropic convolutions. EMMA takes advantage of several independently trained architectures. In particular, EMMA combined DeepMedic, FCN and U-net models and ensembles their segmentation predictions. During training a batch size of 8 was used, and a 3D patch size of 64x64x64. EMMA's ensemble of different models demonstrated a good generalization performance winning the BraTS 2017 challenge. Wang et al. (2017) [5] second place paper took a different approach, by training 3 networks for each tumor subregion in cascade, with each subsequent network taking the output of the previous network (cropped) as its input. Each network was similar in structure and consisted of a large encoder part (with dilated convolutions) and a basic decoder. They also decompose the 3x3x3 convolution kernel into intra-slice (3x3x1) and inter-slice (1x1x3) kernel to save on both the GPU memory and the computational time.

In the BraTS 2018 challenge, top performing submissions included Myronenko (2018) [6] who won 1st place, Isensee et al. (2018) [7] in the 2nd place, McKinly et al. (2018) [8] and Zhou et al. (2018) [9] who shared the 3rd place. Myronenko (2018) [6] proposed an encoder decoder architecture with asymmetrically large encoder to extract deep image features, and a decoder part that is used to reconstruct dense segmentation masks. They also added a variational autoencoder (VAE) branch to the network to reconstruct the input images jointly with segmentation in order to regularize the shared encoder. Isensee et al. (2018) [7] demonstrated that a generic U-net architecture with a few minor modifications is enough to achieve competitive performance. The authors used an additional training data from their own institution (which yielded some improvements for the enhancing tumor dice). McKinly et al. (2018) [8] proposed a segmentation CNN in which a DenseNet structure with dilated convolutions was embedded in U-net-like network. The authors also introduce a new loss function, a generalization of binary crossentropy, to account for label uncertainty. Finally, Zhou et al. (2018) [9] proposed to use an ensemble of different networks: taking into account multi-scale context information, segmenting 3 tumor subregions in cascade with a shared backbone weights and adding an attention block.

As can be seen from the works of previous years, many of the implementations throughout the years use a patch-based model as well as a cascaded one, just with the use of deeper networks with more intricate features

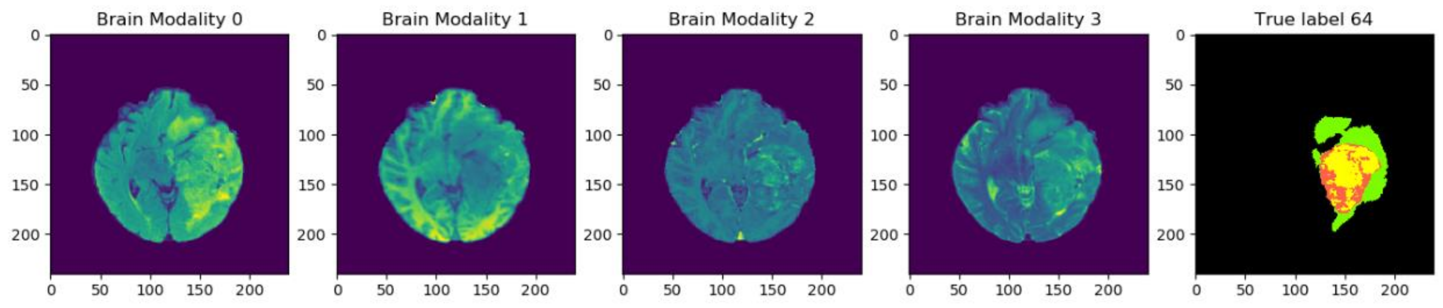

Figure 1 Data of the 4 modalities comprising the brain image channels and the corresponding ground truth mask. Brain modality 0 – FLAIR, Brain modality 1 – T1, Brain modality 2 – T1 enhanced, Brain modality 3 – T2

that have been added throughout the years. We had to use a 2D model instead of a 3D one because of the computational complexity of 3D models and the lack of proper hardware to train the model on.

3. Data

Although we had the BraTS2019 dataset at our disposal, we decided to use the BraTS2013 dataset (which is part of the BraTS 2019 dataset) so we could compare our results to the original article that we were implementing. The training data in the BraTS2013 dataset is comprised of 30 brains all with pixel-accurate ground truth (20 high grade and 10 low grade tumors). For validation, we used several brains from the BraTS2019 training dataset, where each pixel also came with a pixel-accurate ground truth. For each brain there exists four modalities – T1, T1C, T2 and Flair which are co-registered. Each brain comes with a ground-truth for which four segmentation labels are provided, namely non-tumor (class 0), necrosis (class 1), edema (class 2) and enhancing tumor (class 4). In the past, there used to be a non-enhancing tumor (class 3) label as well, but it was removed (is now part of the edema region).

In the BraTS2019 dataset, the data of each patient is located in a separate folder as 5 separate "NIFTI" (Neuroimaging Informatics Technology Initiative) files, one for each of the modalities and another for the segmentation map. Each file is comprised of 155 2D images (slices) of size 240x240. The 4 separate modalities are concatenated to create an MR image with dimensions 155x240x240x4 and 155x240x240x1 for the segmentation labels.

The BraTS dataset is highly imbalanced. Out of all the voxels across the 30 patients (both HGG and LGG), 98.81% are of class 0 (non-tumor tissue or background), 0.28% are of class 1 (necrotic), 0.68% are of class 2 (edema) and 0.22% are of class 4 (enhanced tumor). This makes the segmentation task a difficult one and it is something that we had to take this into account when working on our architecture and methods.

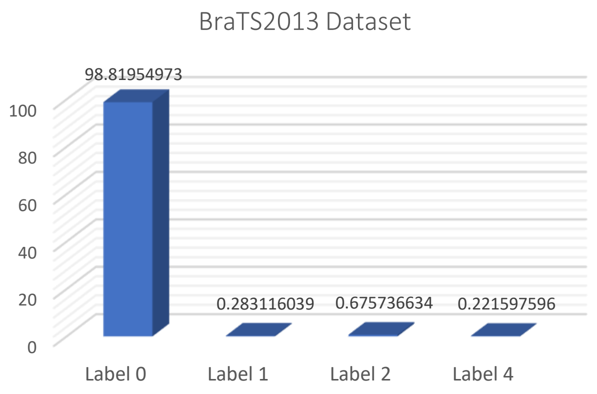

Figure 3 BraTS2013 dataset labels distribution

4. Architectures

The architectures that achieved the best results in the original paper, were all cascaded architectures, composed of a "basic" block which process an $M \times M$ patch and predict the label of its center pixel. For each "basic" block, prediction of the label for the center pixel, of each patch was done without taking into account the labels of neighboring pixels. This is valuable information that should be considered, as the label of a pixel is obviously dependent on its environment.

In order to create a joint segmentation model which will take into account the labels of nearby pixels, the authors used the before mentioned cascaded architectures that feed the output probabilities of a first CNN model as additional inputs to the different layers of a second CNN model. The outputs of the first model are concatenated with different layers of the second model as additional inputs to that layer. The authors proposed that the same CNN block will be used for both models with the difference being the size of the patch that is fed into each model. The "basic" CNN block proposed in the paper was named the "Two-Pathway" architecture and we will go into details of its implementation below.

In the original paper, there were 3 different cascaded networks with the aforementioned architecture, which differ in the location of the concatenation of the output of the first model with the second model. The three architectures are:

1. **InputCascadeCNN** – In this architecture the output of the first model is fed directly as input to the second model and concatenated with its input patches. The additional feature maps simply treated as additional image channels of the input MR image.
2. **LocalCascadeCNN** – In this architecture the output of the first model is concatenated with the first hidden layer of the local pathway stream in the second model.
3. **MFCascadeCNN** – In this last architecture the output of the first model is concatenated to the very end of the second CNN to the layer right before its output layer.

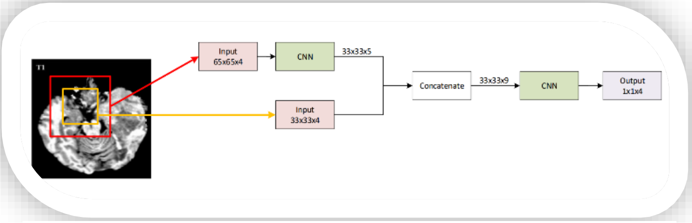

Figure 2 Cascaded architecture with Two-Pathway block(up), Two-Pathway block(down), comprised of global path (red convolution block with MaxOut) and local path (yellow blocks with MaxOut, both are concatenated and a convolution with softmax operation is performed for pixel label prediction

In order to concatenate the output of the first model with one of the layers of the second model, the patch which is fed into the first model must be a lot larger in comparison to the patch that is fed into the second model (the one that outputs the prediction for the pixel in the middle of the patch). For example, in the case of the InputCascadeCNN model, the patch which is fed into the first two-pathway model is of size $65 \times 65 \times 4$. The output of this model is of size $33 \times 33 \times 5$, which is then fed into the second two-pathway model as additional channel inputs.

Because of time and hardware constraints, we did not have time to test the 3 different cascaded architectures and we decided to only implement our own version of the InputCascadeCNN architecture, since it is the one that achieved the best results for the original authors. We implemented our own version of the architecture with some modifications and enhancements to both the architecture and the training procedure. Furthermore, in an effort to improve over the results of the article, we tried a few other architectures which use the same cascaded structure where we replaced the "simple" two-pathway model proposed in the original paper with different models. The main goal was to add more depth or width to the network, as the original network was very shallow (10 convolutions deep), and as we know, in most cases, deeper is better.

We will elaborate about our implementation of the "Two-Pathway" architecture and follow it by the description of our proposed alternatives.

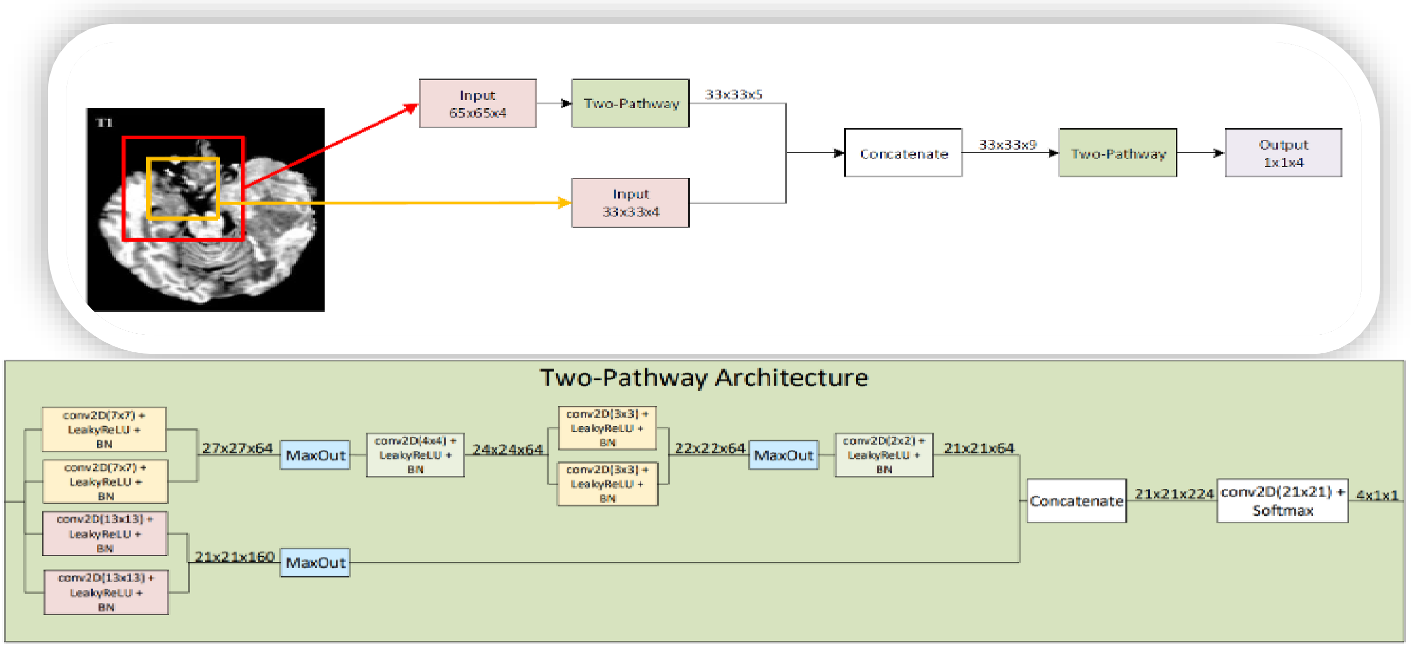
T1
Input 65x65x4
Two-Pathway
33x33x5
Input 33x33x4
Concatenate
33x33x9
Two-Pathway
Output 1x1x4
Two-Pathway Architecture
conv2D(7x7) + LeakyReLU + BN
conv2D(7x7) + LeakyReLU + BN
27x27x64
MaxOut
conv2D(4x4) + LeakyReLU + BN
24x24x64
conv2D(3x3) + LeakyReLU + BN
conv2D(3x3) + LeakyReLU + BN
22x22x64
MaxOut
conv2D(2x2) + LeakyReLU + BN
21x21x64
conv2D(13x13) + LeakyReLU + BN
conv2D(13x13) + LeakyReLU + BN
21x21x160
MaxOut
Concatenate
21x21x224
conv2D(21x21) + Softmax
4x1x1

Figure 5 Cascaded model. In red the 65x65x4 large patch, in yellow the 33x33x4 patch. The CNN block in yellow is the generic part of the model and is replaced with different blocks of architectures

4.1. Two-pathway Block

The basic building block of the Input-Cascade architecture is a simple design consisting of several convolutional layers named the Two-Pathway architecture. The architecture is composed of two streams, a local pathway and a global pathway. The local pathway is composed of a 7×7 convolution layer followed by a 3×3 convolution layer. The global pathway is composed of a single 13×13 convolution layer. The motivation behind this architecture 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", giving some context to where the patch is located in the brain. The 3×3 convolution and polling layers which make up the local pathway is there to give the local pathway the same spatial dimension as that of the global pathway, which will allow us to concatenate the two. The concatenated outputs of these two paths are fed into another convolution layer which is then fed to the softmax output layer which gives the normalized probabilities for the patch's center pixel out of the 4 label classes (0-non tumor, 1-necrotic, 2-edema, 4-enhanced tumor).

The number of parameters in the original model was relatively small. We decided that we would like to add more parameters so the network will be able to learn the details of the brain in a better way. For that reason, we have made some alterations to the original design. We replaced the pooling layers with stride 1 that were used in the original design with convolution layers.

Furthermore, in order to implement the maxout operation, we use two convolutions that receive the same input and takes the maximum of their outputs, thus implementing a maxout between the two. During training we experienced a condition where layers began to output zeros, which resulted in Nan loss value. We found out that changing the activation function from ReLU to LeakyReLU solved the backprop problem because the network also learns when values are negative. We added batch-normalization layers following every convolution layer and decided to only use l2 regularizations with smaller weights instead of the L1-L2 regularizations proposed in the paper.

Our implementation of the cascaded architecture with the Two-pathway block contained 1,991,997 parameters and a had a relatively shallow depth of 10 layers.

4.2. Custom Res-net Block

A Custom Res-net block composed of a Res-net like cell that uses a mixture of "valid" (no zero padding) and "same" (with zero padding) convolutions, along with our own version of a skip connection. This block does not contain any skip connections as the ones that appear in the original Res-net architecture. That is because we cannot use pooling layers in the architectures because of the way inference is implemented (more details under the inference section).

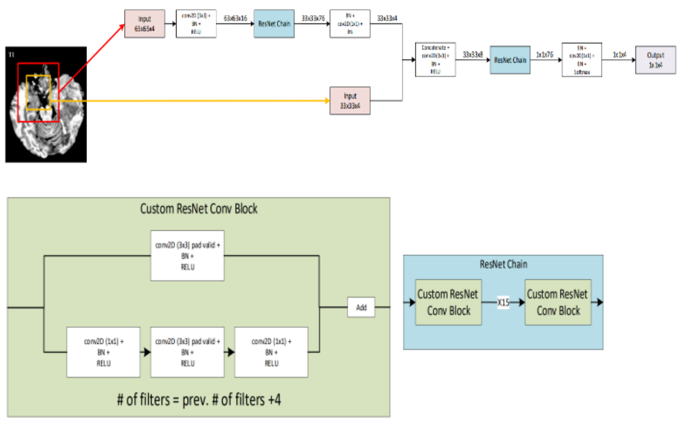

Figure 4 Cascaded RenseNet architecture (up) the RenseNet Chain outputs 76 feature maps deep followed by a 1x1 convolution for adaptation to cascaded architecture. RenseNet Chain (down right) is comprised of 15 RenseNet blocks (down left)

The cascaded architecture with the custom Resnet block contained 1,422,840 parameters and had a depth of 94 layers.

4.3. Custom Dense-net Block

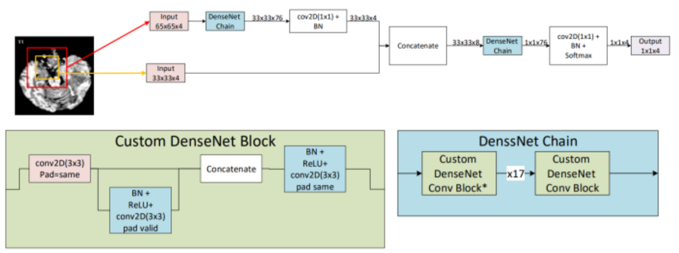

Figure 6 Cascaded DenseNet architecture (up) the DenseNet Chain outputs 76 feature maps deep followed by a 1x1 convolution for adaptation to cascaded architecture. DenseNet Chain (down right) is comprised of 17 DenseNet blocks (down left)

A Custom Dense-net block composed of a Dense-net like cell that uses a mixture of "valid" (no zero padding) and same (zero padding) convolutions, along with our own version of a skip connection. As in the case of the Res-net custom block, this block does not contain any skip connections because of inference related reasons.

The cascaded architecture with the custom Densenet block contained 1,278,656 parameters and had a depth of 102 layers.

4.4. Three-pathway Block

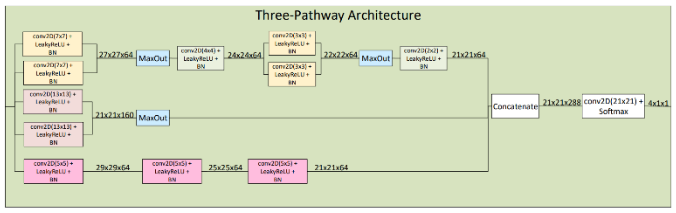
Three-Pathway Architecture
conv2D(7x7) + LeakyReLU + BN
conv2D(7x7) + LeakyReLU + BN
27x27x64
MaxOut
conv2D(4x4) + LeakyReLU + BN
24x24x64
conv2D(3x3) + LeakyReLU + BN
conv2D(3x3) + LeakyReLU + BN
22x22x64
MaxOut
conv2D(2x2) + LeakyReLU + BN
21x21x64
conv2D(13x13) + LeakyReLU + BN
conv2D(13x13) + LeakyReLU + BN
21x21x160
MaxOut
conv2D(5x5) + LeakyReLU + BN
29x29x64
conv2D(5x5) + LeakyReLU + BN
25x25x64
conv2D(5x5) + LeakyReLU + BN
21x21x64
Concatenate
21x21x288
conv2D(21x21) + Softmax
4x1x1

Figure 7 Three-pathway architecture incloding the 2 original global path and local path with the addition of the more local path

Similar to the Two-pathway block, this block uses the same global and local paths and also adds a third path in parallel to these two with three 5×5 convolution layers, each followed by a LeakyReLU activation layer and a batch normalization layer. The third path was supposed to examine the details in the environment that is even closer to the center pixel of the patch and provide additional channels and more parameters to the network in an effort to make it wider and enable it to learn finer details. Here to the reason for adding three 5x5 convolution is due to the inference method.

The cascaded architecture with the Three pathway block contained 2,678,333 parameters and had a depth of 10 layers.

5. Methods

5.1. Training

As mentioned before, the data in the BraTS2013 dataset is highly imbalanced. Since the training is based on patches, selecting patches from the true distribution in the dataset would cause the model to be overwhelmed by healthy patches which will be a problem when training out CNN models. To combat this imbalance, in the original paper, a 2-phase training procedure was used for both the “basic” block (the Two-pathway block) and the cascaded architecture (InputCascadeCNN).

For the training of the Two-pathway block, in the first phase, a patch dataset was constructed such that all labels are equiprobable (the number of patches of each class is equal, where a patch is identified by the label value of its center pixel). The model is then trained on the equiprobable patches giving equal weights to the 4 labels which results in false positives of the tumor related labels because of the enhancement in distribution that they were given. This was done in order to account for the imbalance in the dataset. During the second phase of training, all the weights of the model are frozen and only the output layer is trained with a more representative distribution of the labels by randomly selecting patches from within the different brains. Using the two-phase training results in the best of both worlds – the bulk of the parameters in the model are set in a balanced way during the first training phase and the outputs are then recalibrated during the second phase to account for the true distribution of the dataset.

For the training of the InputCascadeCNN architecture, the authors started by training the Two-pathway block with the two-phase procedure described above. Then, they fixed the parameters of the Two-pathway block and include it in the cascaded architecture and move to training the remaining parameters using a similar procedure. The numbers of patches used according to the article was 2.2 million and 3.2 million for the first and seconds phase respectively.

We initially tried to implement the training procedure that was proposed in the article but we noticed that it takes too long and did not provide us with good results for each separate phase (the results using the Two-pathway CNN should have been close to the results with the cascaded CNNs). As a result, we instead decided to use an end-to-end training procedure of the cascaded architecture with the selected “simple” block as part of it. This saved a lot of time in the training procedure and let us view the results of our entire model in real time, without having to train the different pieces of the architecture first.

5.2. Pre-processing

We applied the following pre-processing steps for all the data in the training data set:

1. Applied N4ITK bias field correction. This is a bias field correction algorithm that is used to correct areas in the MR image that are brighter than their surrounding due to a fault in the MR machine.
2. Removed the 0.025% of the highest intensity voxels across each channel in each brain. As mentioned before, voxel values in MR images are not normalized, and therefore not bounded, and may have very large values as well as negative values. In the BraTS dataset the minimum value is always 0 (background) but the maximal voxel value can be very large. We clipped the highest intensity voxels to remove outliers. In the original paper the authors removed 1% of the highest and lowest intensities. We noticed that removing 1% causes many pixels to be removed, making the image much brighter while losing many of the details in the image so we tried different values until we settled on this one.
3. Normalizing each modality per each brain so its voxel values will be in the range of 0 to 1. This was done so all the data will have the same scale to prevent exploding gradients during training.
4.

5.3. Loss

In the original paper, the authors used categorical cross entropy as the loss function of choice and used the 2-phase training we have elaborated about earlier. In order to make the training process simpler and faster, we also used the equiprobable approach for the selection of all of the patches from the 4 pixel labels, but instead of using the categorical cross-entropy loss function, we decided to try a few other loss functions. The loss function that we ended up using is the sum of the **Jaccard loss** (intersection over union) and the **categorical cross entropy** loss. We will explain about the different loss functions below:

1. **Categorical Cross-Entropy Loss** – The true class is represented as a one-hot encoded vector, and the closer the model’s outputs are to that vector, the lower the loss. In our case, we are looking at the label of the pixel in the center of each patch.

$$L(y, \hat{y}) = -\sum_{i=0}^{C} y_i \log(\hat{y}_i)$$

$\hat{y}$ – predicted value
y – ground truth, where the probability of the true class is set to 1 and 0 for the other classes.

2. **Weighted Categorical Cross-Entropy Loss** – The true label of the center pixel of each patch is multiplied by a weight value which is equal to the reciprocal proportion of the number of pixels of that class in the entire dataset. We normalize the weight such that the weight of the largest class will be 1. This way we account for the great imbalance in the dataset. The motivation is to punish mistakes in the pixels which belong to classes which are less abundant by multiplying them by larger weights because of their distribution and thus “encouraging” the network not to label those wrongly, since it will result in a large loss.

$$L(y, \hat{y}) = -\sum_{i=0}^{C} W_i y_i \log(\hat{y}_i)$$

$\hat{y}$ – predicted value
y – ground truth, where the probability of the true class is set to 1 and 0 for the other classes.
W_i – Weight of each class.

3. **Dice Loss** – The Dice coefficient is used for measuring the similarity between two sets, or in our case, the true labels and the predicted labels for a given dataset. It is widely used in segmentation problems and is the metric which is used for evaluating the results of the BraTS challenge. It is given by:

$$DSC = \frac{2TP}{2TP + FP + FN}$$

Where TP, FP, FN are True positive, False Positive and False Negative. It ranges between 0 and 1, where it is equal to 0 when all the samples are wrongly classified (sets are completely disjoint) and 1 when all samples are correctly classified (perfect match). The Dice score works well for imbalanced datasets because it measures relative overlap between the prediction and ground truth, without being affected by the overall number of pixels from each class (as in the case of categorical cross entropy). The Dice loss is defined as:

$$L_{Dice} = 1 - DSC$$

4. **Weighted Dice Loss** – A Dice loss where each sample is multiplied by a weight. The weight is defined in the same manner as explained for the Weighted Categorical Cross Entropy loss.

5. **Jaccard Loss** – Also known as the Intersection Over Union, the Jaccard index measures the similarity between two sets, similar to the Dice coefficient, with the only difference being the factor of 2 multiplying the true positive samples in the Dice coefficient.

$$IOU = \frac{TP}{TP + FP + FN}$$

The fact that the true positive samples are not multiplied by a factor of 2 means that the samples which are misclassified (false positive and negative samples) are greatly penalized by the IOU compared to the Dice coefficient. The IOU tends to have a "squaring" effect on the errors relative to the Dice score. Since it also measures the overlap between the prediction set and the ground truth, it is suitable for imbalanced datasets. The Jaccard loss is defined as:

$$L_{Jaccard} = 1 - IOU$$

5.4. *Optimization*

Instead of looking at a single brain on each iteration, we look at a mini batch of brains, where the number of brains is controlled by a hyperparameter. For each brain which is part of this minibatch, we randomly generate an equal number of patches of each label from throughout the brain (the number of patches is also controlled by a hyperparameter). We applied random flips and 90 degree turns to all patches. We then train our model on all the patches, and their corresponding labels, which were generated from all the brains in the minibatch and use stochastic gradient descent as the selected optimization algorithm. We used weight clipping for weights greater than 0.5 as well as momentum with value of 0.9. During training we also tried to use the Adam optimizer nut SGD gave us the best results without being noticeably slower.

It is important to note that for each epoch we randomize the order of brains which are introduced to the model and also the patches which are generated from the brains in the mini batch. This was done to create more variability and present the model with different inputs on each epoch.

We have used a few methods to control the **learning rate** of our model during training:

1. **Scheduled decay** – After experimenting with different values, we ended up selecting an initial learning rate of $lr = 0.0005$ which is used for a certain number of epochs and is then divided by 10. This is the simplest decay method, but it provided us with the best results during our tests.
2. **Cyclic Learning Rate** – We have experimented with a few methods of cyclic learning rate including cosine, triangular and exponential decay. Although we expected CLR to provide us with the best results, it did not surpass those achieved with the simple learning rate decay method. We thought this may have be attributed to the choice of hyperparameters, but we have tried multiple setups and we did not see an improvement using any of the CLR methods. Below are the different

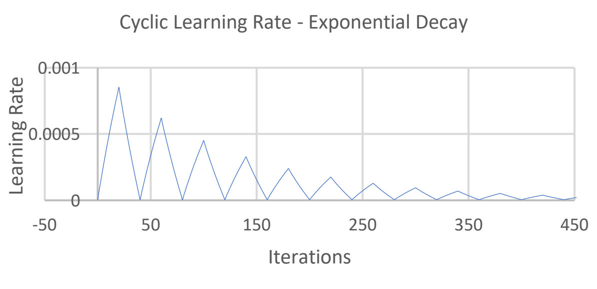

Figure 8 Exponential Decay learning rate procedure

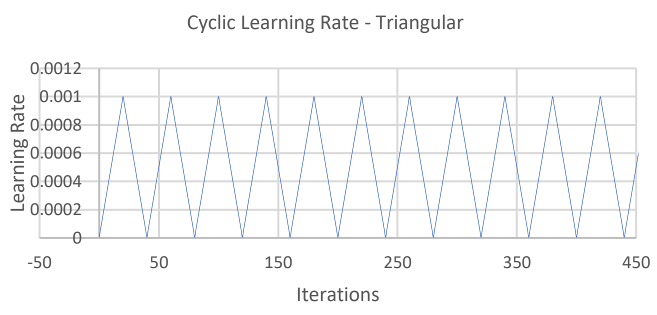

Figure 9 Triangular learning rate procedure

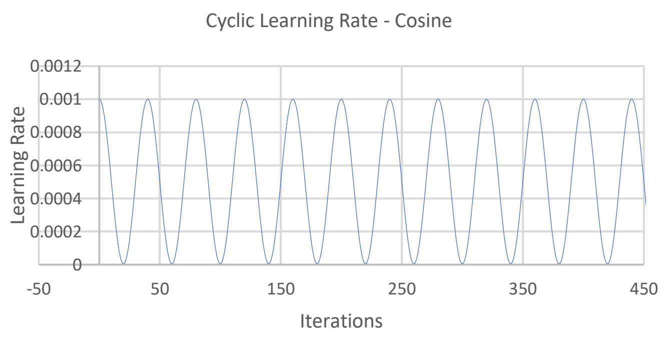

Figure 10 Cosine learning rate procedure

5.5. *Regularization*

We used L_2 weight regularization with a weight of $\lambda_{L_2} = 0.0005$ on the convolutional kernel parameters as well as batch normalization following each convolution layer in the design. We initially tried using both L_1 and L_2 regularizations with the default values of $\lambda_{L_1} = 0.01$ and $\lambda_{L_2} = 0.01$ but did not achieve good results. After reading the papers by Alex Krizhevsky, et al. (2012) [10] and Karen Simonyan and Andrew Zisserman (2015) [11], we noticed that L_1 regularization is not used for CNN's and that smaller values should be used for L_2 regularization. The suggested method in the articles provided us with results that were significantly better relative to using both the L_1 and L_2 regularization and using no weight regularization where we noticed slight overfitting.

For batch normalization, we have used the more recent approach where the batch normalization layer come after the activation layer and not the other way around [12] (as implemented in the original Resnet paper).

5.6. *Inference*

The straightforward way of segmenting a brain using our cascaded model is to divide the entire brain into slices and each slice into patches, feed pairs of small and large patches into the model and then concatenate all the predicted results of our model to receive the full segmentation of each slice of the brain. The predicted label mask for a single slice should be of size 240×240. Because the larger patches are of size 65×65, we will need to zero-pad the image on each side by 32 pixels to eventually receive a prediction with the same size as that of the original image. As a result, the dimensions of each image will become 304×304 voxels from which we will generate 57,600 patches which will be fed into the model for classification of the entire image slice. Following this approach results in inference time of over 15 minutes for a single image using the Nvidia RTX 2060 Super GPU that we were using. The data for each patient is comprised of 155 slices, therefore it is practically impossible to evaluate our model on a validation dataset during training.

As a workaround, we looked at each slice as a large image and fed it into the model for prediction. This means that in order to run inference on an entire brain we only need to go over all the slices, with one pass over each slice. We made sure that both methods (patch oriented and full slices) return the same results. This result showed us that even though the training was done on single voxels at the center of patches and not full images, the model learns the features

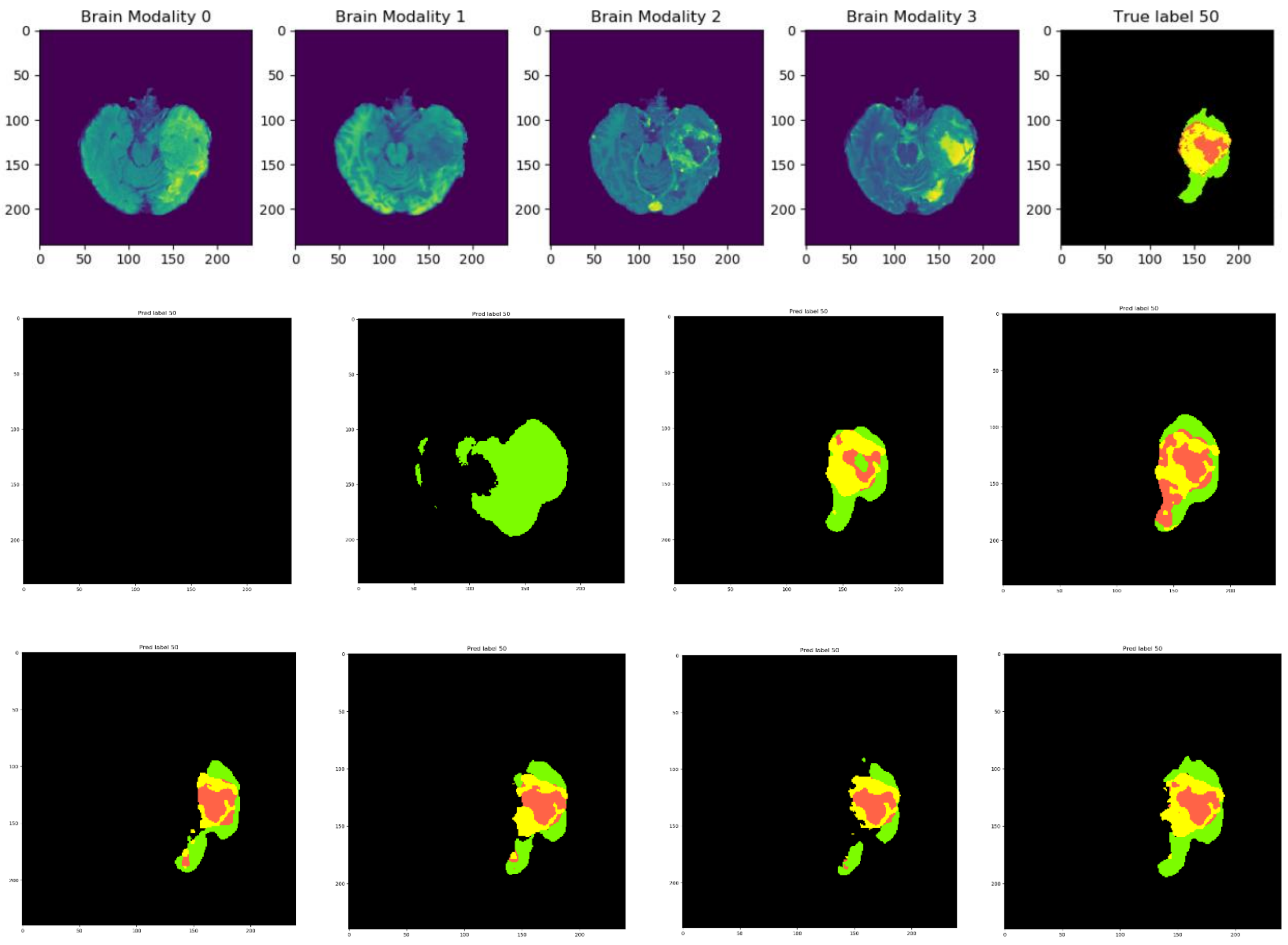

Figure 11 Learning progression of the InputCascadeCNN architecture. The first row shows a chosen slice with its 4 modalities and ground truth segmentation mask(right). The stream of figures on the second and third rows, from left to right show the learning process of the network on each improvement it has made. As the model learns better features, it can better distinguish the boundaries between the tumor sub-classes.

of the brain and can make predictions based on entire images. Using this solution, we were able to evaluate our results during training and save the best weights based on the validation data set. Since the model is eventually evaluated using the Dice score, on each validation stage we predicted the label of each voxel for all the brains in the validation dataset and measured the Dice score for each label class. We looked at the average Dice score of the tumorous classes – 1 (necrotic), 2 (edema) and 4 (enhancing tumor) and saved the weights based on improvements in the metric.

6. Experiments

As we explained in details in the previous sections, we used the BraTS 2013 dataset to train the model, which is comprised of 30 patients (20 HGG and 10 LGG). We select random brains to create a mini batch of brains from within the dataset, and from each brain we select an equal number of patches, randomly, from each class, apply preprocessing and augmentations and then train the model on these patches using stochastic gradient descent.

As can be seen from the graph below, the model that achieved the best results was the cascaded model with our implementation of the Two-pathway block. The other models, that were either deeper or wider than the Two-pathway model, were a lot slower to train and achieved poor results in comparison.

We thought the Renset and Densenet models would achieve better result compared to the Two-pathway model because of their increased depth and their improved performance in classification tasks. Because of the inference method we used, where an entire slice is fed into the model instead of patches, we could not use pooling layers with stride 2. Using pooling layers would result in different image size at the output, depending on the input image size and therefore we resorted to only using valid convolutions that will decrease the image size. Using only convolutions meant that we cannot use a proper skip connection as the ones used in the original Resnet and Densenet architectures and we had to place a convolution layer on that path, which as a result didn't provide us with the "highway" that is used during back propagation and which gives such architectures their advantage. We believe this hurt our results and prevented the improvement of those architectures.

The Three-pathway architecture provides more width in comparison to the Two-pathway architecture, by adding the path with the 5×5 convolutions. We though the additional channels and parameters that were added on top of the Two-pathway model would result in improved results or ones that are at least on par with those of the Two-pathway model. Instead, the added parameters resulted in poor results and slower training. In this case, we believe that better selection of hyper parameters of the additional path should have improved our results.

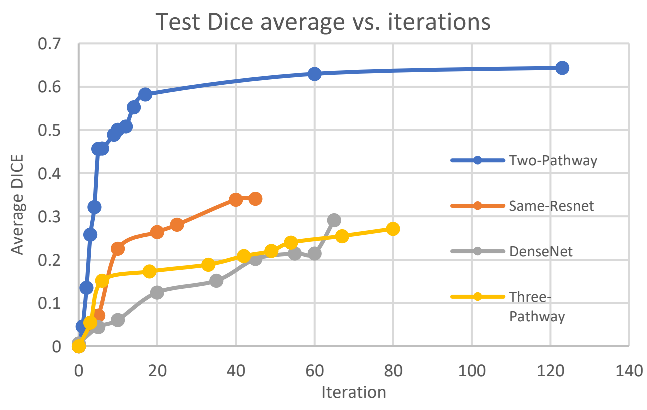

Figure 12 Average Dice score of the different architectures

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
Our Two-Path Cascaded	0.86	0.80	0.72	1	1	1	0.81	0.78	0.77

Figure 13 Comaprison of our implemented architecture to the architecture that achieved the best results in the original paper

Figure 12 shows the progression of each of the architectures that we implemented. The points that appear on the graph are only the iterations on which the network has made an improvement in the average Dice score over the 3 tumorous classes. We ran all the architectures for the same number of iterations but the Resnet, Densenet and Three-pathway based architectures stopped improving at an early stage while the Two-pathway based model kept improving at later iterations as well. It can be clearly seen that the results that were achieved using the Two-pathway model were substantially better than all the other architectures.

The quantitative evaluation of the model is done using 6 randomly selected brains from the BraTS 2019 dataset. The model is evaluated according to 3 different tumor regions (mainly due to practical clinical applications) which are defined as:

1. The complete tumor region (including all three tumor structures).
2. The core tumor region (including all tumor structures except “edema”).
3. The enhancing tumor region (including the “enhanced tumor” structure).

For each tumor region, Dice, Sensitivity and Specificity are computed as follows:

$$Dice = \frac{2TP}{2TP + FP + FN}$$
$$Sensitivity = \frac{TP}{TP + FN}$$
$$Specificity = \frac{TN}{TN + FP}$$

Where:
$TP = True\ Positive, TN = True\ Negative,$
$FP = False\ Positive, FN = False\ Negative$

During the initial training of the different architectures, we used a plotting function to view the output of our model and compare it to the ground truth. This gave us a glimpse into the performance of our algorithm which helped us make the necessary changes to our architecture and methods. Below are the outputs of the cascaded architecture using the Two-pathway CNN block which achieved the best results. We saved the model’s weights during its training on each iteration where it improved on the average Dice score over the tumor classes. We show the network’s prediction on each improvement that was made. It is worth mentioning that the average Dice score was computed over all the brains in the validation dataset, so an improvement in the overall Dice score may not translate into an obvious visual improvement in terms of the predicted mask of a single slice from a single brain.

In the table above we compared our results with the cascaded architecture which contains the Two-pathway model to the InputCascadeCNN architecture presented in the original paper. As can be seen from the table, our results are very close to those achieved in the original paper.

One thing that popped in the results is that the calculated specificity of our model is 1 for all 3 tumor regions. This can be attributed to the true negative samples which are calculated over all 4 classes including class 0 which is comprised of non-tumor pixels (background and healthy pixels). Since these pixels make up 99% of the data, even after taking into account the pixels which are misclassified, the calculated specificity still approaches 1. We could not find the exact way specificity was calculated in the official BraTS challenge, so we used the straightforward calculation according to the specificity formula over all the voxels in the validation dataset.

The prediction time using our model is approximately 15 seconds per brain, which is $1\frac{2}{3}$ times faster than the Two-pathway model (standalone) used in the original paper and 12 times faster than the cascaded architecture using the Two-pathway model.

7. Conclusion

In this work, we presented a patch-based semantic segmentation network for brain tumor segmentation from multimodal 3D MRIs, based on the article “Brain tumor segmentation with Deep Neural Networks”. The proposed cascaded architecture uses two instances of the same “Two-pathway” module where the output of the first module is fed as additional input channels into the second model.

We tried to use various architectures, training methods including different hyperparameters, learning rate controls, optimizers and regularizations but eventually the cascaded architecture comprised of the “Two-pathway” model achieved the best results.

We do believe that a similar model which uses 3D convolutions over several adjacent slices instead of only using 2D data would result in additional improvement and it is something we would have liked to try if we had stronger hardware at our disposal.

Our proposed cascaded architecture works very fast and segments an entire brain in less than 15 seconds. This makes our method practical for brain segmentation and may provide a quick, reliable and accurate solution which may aid radiologists in their work.

References

[1] Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, Hugo Larochelle: Brain Tumor Segmentation with Deep Neural Networks. In: Medical Image Analysis 35 18–31 (2017)

[2] Zikic, D., Ioannou, Y., Brown, M., Criminisi, A.: Segmentation of brain tumor tissues with convolutional neural networks. Proc. In: BRATS-MICCAI (2014)

[3] Urban, G., Bendszus, M., Hamprecht, F., Kleesiek, J.: Multi-modal brain tumor segmentation using deep convolutional neural networks. Proc. In: BRATS-MICCAI (2014)

[4] Kamnitsas, K., W. Bai, E.F., McDonagh, S., Sinclair, M., Pawlowski, N., Rajchl, M., Lee, M., Kainz, B., Rueckert, D., Glocker, B.: Ensembles of multiple models and architectures for robust brain tumour segmentation. In: International Conf. on Medical Image Computing and Computer Assisted Intervention. Multimodal Brain Tumor Segmentation Challenge (MICCAI, 2017). LNCS

[5] Wang, G., Li, W., Ourselin, S., Vercauteren, T.: Automatic brain tumor segmentation using cascaded anisotropic convolutional neural networks. In: International Conf. on Medical Image Computing and Computer Assisted Intervention. Multimodal Brain Tumor Segmentation Challenge (MICCAI, 2017). LNCS

[6] Myronenko, A.: 3D mri brain tumor segmentation using autoencoder regularization. BrainLes 2018, Springer LNCS 11384 (2019) 311–320

[7] Isensee, F., Kickingereder, P., Wick, W., Bendszus, M., Maier-Hein, K.H.: No newnet. In: International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI 2018). Multimodal Brain Tumor Segmentation Challenge (BraTS 2018). BrainLes 2018 workshop. LNCS, Springer (2018)

[8] McKinley, R., Meier, R., Wiest, R.: Ensembles of densely-connected cnns with label-uncertainty for brain tumor segmentation. In: International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI 2018)

[9] Zhou, C., Chen, S., Ding, C., Tao, D.: Learning contextual and attentive information for brain tumor segmentation. In: International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI 2018). Multimodal Brain Tumor Segmentation Challenge (BraTS 2018). BrainLes 2018 workshop. LNCS, Springer (2018)

[10] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In: NIPS (2012)

[11] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In: ICLR (2015)

[12] Evaluation of BatchNorm layer performance on ImageNet-2012