February 2018

Automatic Brain Tumor Segmentation by Deep Convolutional Networks and Graph Cuts

Zhenyi Wang
The University of Western Ontario

Supervisor
Veksler, Olga
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Master of Science

Recommended Citation

Wang, Zhenyi, "Automatic Brain Tumor Segmentation by Deep Convolutional Networks and Graph Cuts" (2018). *Electronic Thesis and Dissertation Repository*. 5189.
https://ir.lib.uwo.ca/etd/5189

Abstract

Brain tumor segmentation in magnetic resonance imaging (MRI) is helpful for diagnostics, growth rate prediction, tumor volume measurements and treatment planning of brain tumor. The difficulties for brain tumor segmentation are mainly due to high variation of brain tumors in size, shape, regularity, location, and their heterogeneous appearance (e.g., contrast, intensity and texture variation for different tumors). Due to recent advances in deep convolutional neural networks for semantic image segmentation, automatic brain tumor segmentation is a promising research direction.

This thesis investigates automatic brain tumor segmentation by combining deep convolutional neural network with regularization by a graph cut. We investigate several deep convolutional network structures that have been successful in semantic and medical image segmentation. Since the tumor pixels account for a very small portion in the whole brain slice, segmenting the tumor from the background is a highly imbalanced dense prediction task. We use a loss function that takes the imbalance of the training data into consideration. In the second part of the thesis, we improve the segmentation results of a deep neural network by using optimization framework with graph cuts. The graph cut framework can improve segmentation boundaries by making them more smooth and regular. The main issue when using the segmentation results of convolutional neural networks for the graph cut optimization framework is to convert tumor probabilities learned by a convolutional network into data terms. We investigate several possible ways that take into consideration the segmentation artifacts by convolutional neural networks.

In experiments, we present the segmentation results by different deep convolutional neural network structures, e.g., fully convolutional neural network, dilated residual network and U-Net. Also, we compare the combination of U-Net with different data terms for graph cut regularization to improve the neural network segmentation results. Experimental results show that the U-Net performs best with the intersection over union (IoU) for tumors of 0.7286. The IoU for tumors is improved to 0.7530 by training on three slices. Also, the IoU for tumors is improved to 0.7713 by U-Net with balanced loss function. The IoU for tumors is further improved to 0.8078 by graph cut regularization.

Keywords: Automatic brain tumor segmentation, deep convolutional neural network, dense prediction, graph cut regularization.

Acknowledgement

First of all, I would like to express my sincere gratitude to my supervisor Prof. Olga Veksler who is a great and very nice advisor and inspires me to apply deep learning model to solve computer vision problems. I really appreciate her patience when discussing the next step research directions and looking through the experimental results. And every time when we get stuck she can always come up with new ideas to help us move forward. Her attitude and dedication to work and research encouraged me to do research carefully.

Second, I want to appreciate those helped me with my study and thesis. I am so grateful to work with Dr. Lena Gorelick who helped me to incorporate the deep convolutional neural network output into the energy minimization framework. And I would also like to thank Prof. Yuri Boykov, who has great passion about computer vision and presented very detailed and intuitive explanation in class. His passion and teaching style enabled me to think actively about computer vision research problems and made me realize that computer vision is so interesting.

Next, I would like to thank other members in our vision group. I want to give special thanks to Danfeng Chen who did the prior work of this thesis. She always patiently answers my question about previous research work. Thanks all my other friends as well, they bring much fun to my life. Last but not the least, I want to thank my family, especially my great parents and sister, for continuously encourage me to solve problems, support and sincere love.

Contents

List of Figures

List of Tables

List of Appendices

Chapter 1

Introduction

Brain tumors can be classified into two types: cancerous tumors and benign tumors. Magnetic resonance imaging (MRI) provides detailed images of the brain, and is one of the most common ways used to diagnose brain tumors. Brain tumor segmentation from MRI is helpful for improved diagnostics, growth rate prediction and treatment planning for brain tumors and is crucial for monitoring tumor growth or shrinkage in patients during therapy. Also, it is important in surgical planning or radiotherapy planning. In these cases, the tumor has to be outlined and surrounding healthy tissues are also of interest for further treatment and processing.

Fast and accurate segmentation of a brain tumor is a non-trivial task. The difficulties for brain tumor segmentation are mainly due to high variation of brain tumors in size, shape, regularity, location, and their heterogeneous appearance (e.g., contrast, intensity and texture variation for different tumors).

Previous brain tumor segmentation approaches can be classified into two categories: interactive segmentation [35, 9, 29, 17, 19] and automatic segmentation [16, 62, 53, 1, 25, 40, 47, 2, 8]. Interactive segmentation does not rely on a large training data set. It requires user input, e.g., specifying object of interest or adding seeds indicating the labels of some pixels to belong to certain class. The interactive segmentation also allows users to evaluate the result. Then, users edit and refine the results by adding more seeds if the segmentation results are not satisfactory. Interactive segmentation process can be repeated until no further modification is needed. Although interactive segmentation can obtain good segmentation results, trained experts still need to put much effort into performing segmentation until they get satisfactory results. Hence, this thesis aims to investigate automatic brain tumor segmentation by combining deep convolutional neural networks with graph cut regularization to help automatically obtain tumor segmentations.

This thesis utilizes a previously successful interactive brain segmentation system [35, 9], but does not require users to provide input. This thesis shows how to segment the tumor for the whole slice automatically.

1.1 MRI and Brain Tumor

This thesis uses MRI-based image data. A magnetic resonance imaging (MRI) scan is a radiology technique that uses magnetism, radio waves, and computers to form pictures of the anatomy and the physiological processes of the body in both healthy and non-healthy tissues [11]. MRI can provide a very detailed description of organs and detect tiny body structure changes of the human body. Detailed MRI allows physicians to evaluate various body parts and determine whether certain diseases are present or not.

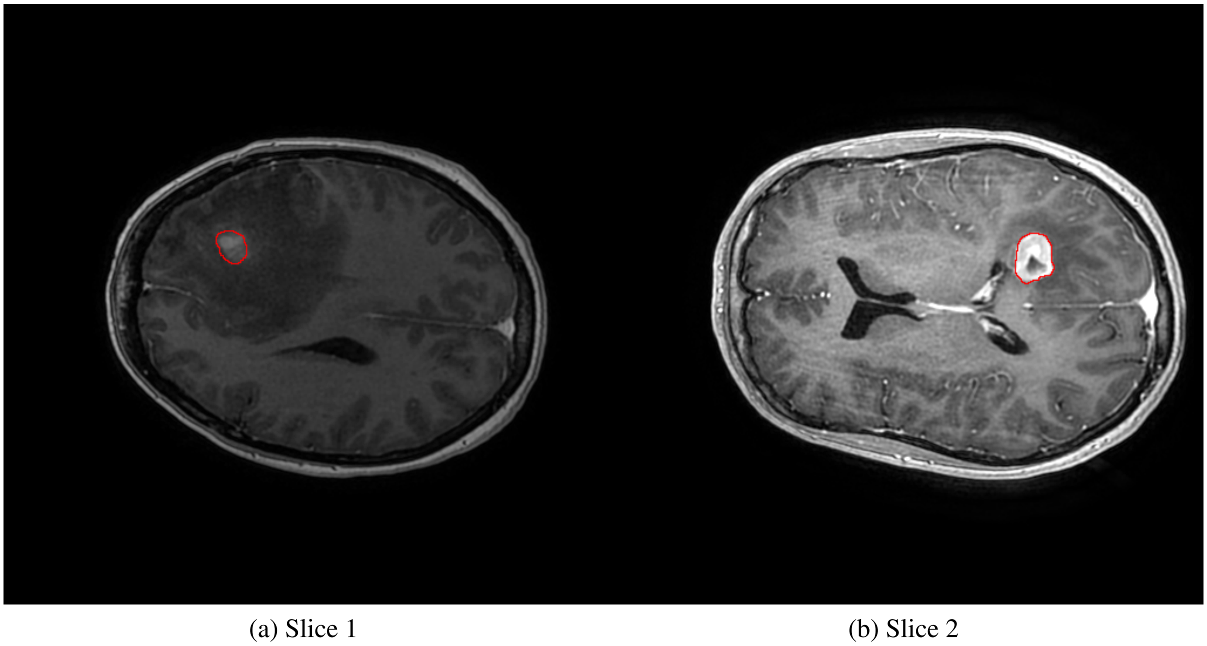
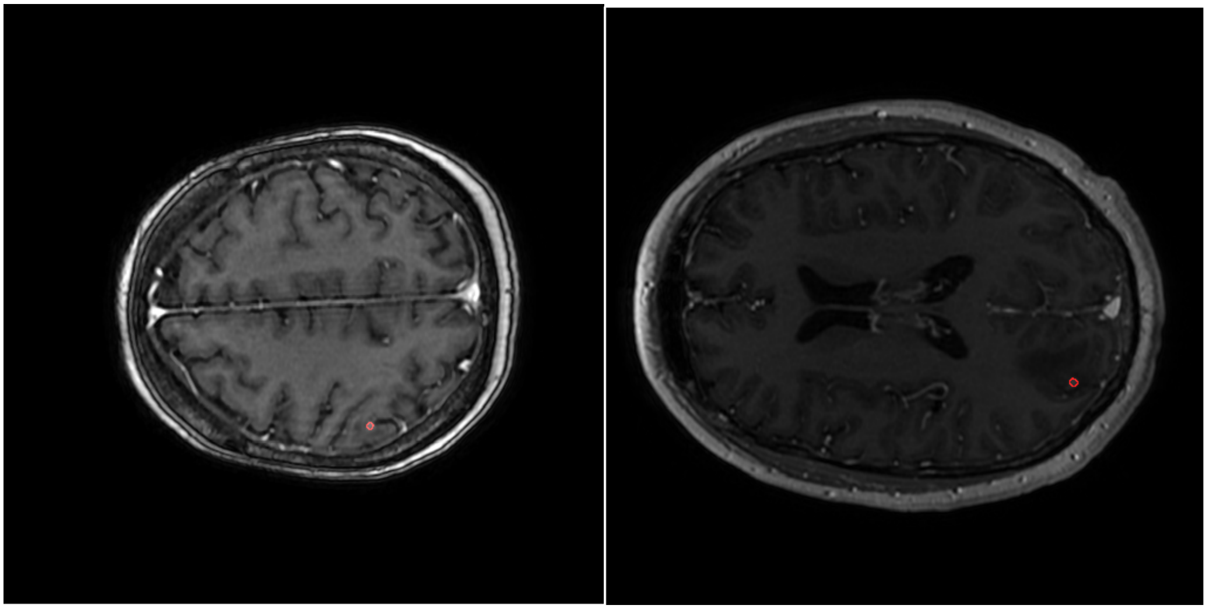

(a) Slice 1 (b) Slice 2

Figure 1.1: Slices of brain image. The red contours are the boundaries of the ground truth. The pixels inside the red contour are tumor pixels and the pixels outside the red contour are background pixels.

In general, the MRI of a brain is a 3D scan or a sampling of the human brain structure in 3D, i.e., x, y and z, spatial dimensions. But the sampling rates can be different along the three different dimensions. That is to say, voxels of the brain scan are not equally spaced along different dimensions. This property is different from traditional images where the distance between neighbouring pixels is assumed to be the same in all different dimensions. Figure 1.1 shows two slices of a brain scan from different patients. An alternative way to think about the 3D MR image is a 3D volume composed by a series of slices in one fixed direction. The intensity of a voxel is proportional to the nuclear magnetic resonance signal intensity of the corresponding volume element or voxel being imaged. The dimension of individual MR slice image determines the spatial resolution, or the level of detail that can be observed in one slice image. The dimension in each direction may vary from one volume to another, depending on many factors, including imaging parameters, magnet strength, the time allowed for acquisition, etc.

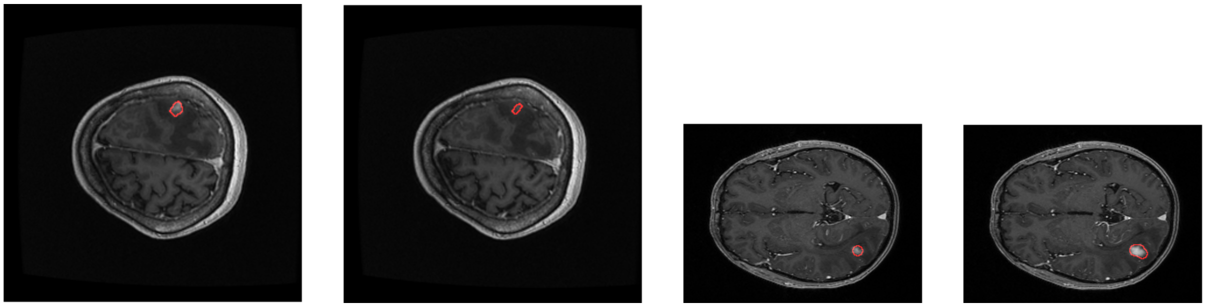
Our data is provided by Aaron Ward and Glenn Bauman from Lawson Health Research Institute. There are 64 MRI brain scans from 27 patients. Some scans are from the same patient in different stages. Each volume has a ground truth, where each pixel is labeled as tumor or background, provided by Glenn Bauman, who is a radiation oncologist and London Regional Cancer Program Professor. Some ground truth is imprecise because it was obtained by blob-manipulation tools. We remove some volumes. Then, we have 45 volumes. Some volumes are deleted for several reasons. First, tumors in some volumes are too small. If we add these volumes into training set, it will cause neural network to fit these small tumors. When making predictions on test data, the neural network will generate false positives. Second, the slice resolution of some brain volumes is too small compared to that of other brain volumes. Significantly resizing these volumes to larger size will change original information. Third, some volumes are too dark and different from most of other volumes. Finally, some volumes do not have voxels that are labeled as tumors. Figures 1.2, 1.3 and 1.4 show some examples of removed volumes.



(a) Slice 1

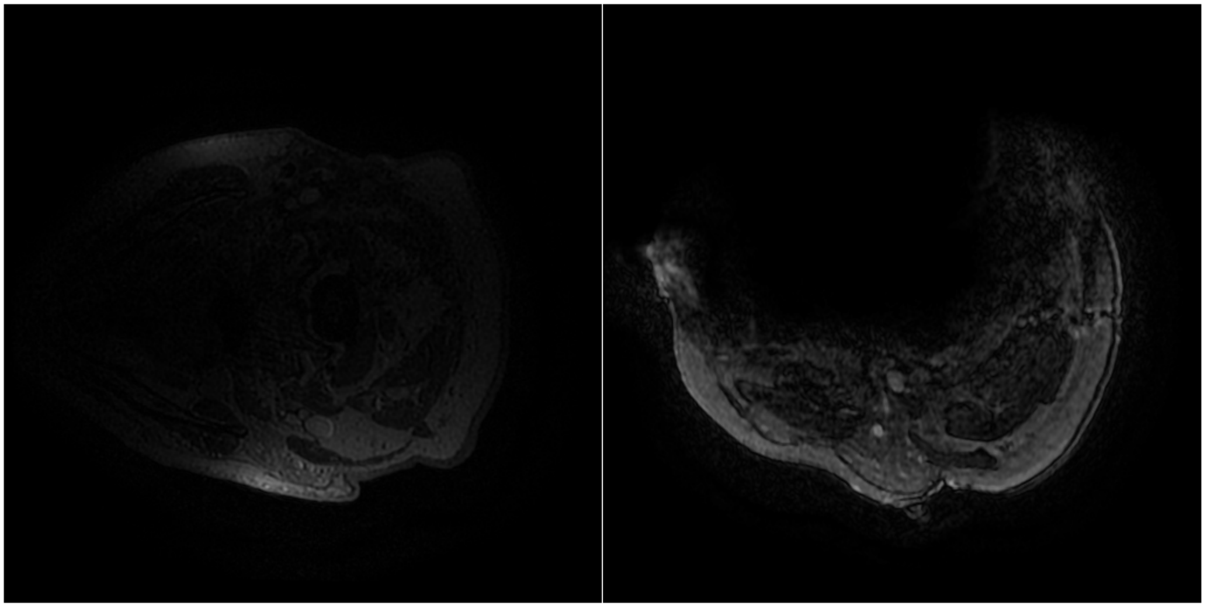
(b) Slice 2

Figure 1.2: Tumor size is too small. The red contours are the boundaries of the ground truth. The pixels inside the red contour are tumor pixels and the pixels outside the red contours are background pixels.



(a) Slice 1 (b) Slice 2 (c) Slice 3 (d) Slice 4

Figure 1.3: Volume resolution is too small. The red contours are the boundaries of the ground truth. The pixels inside the red contours are tumor pixels and the pixels outside the red contours are background pixels.



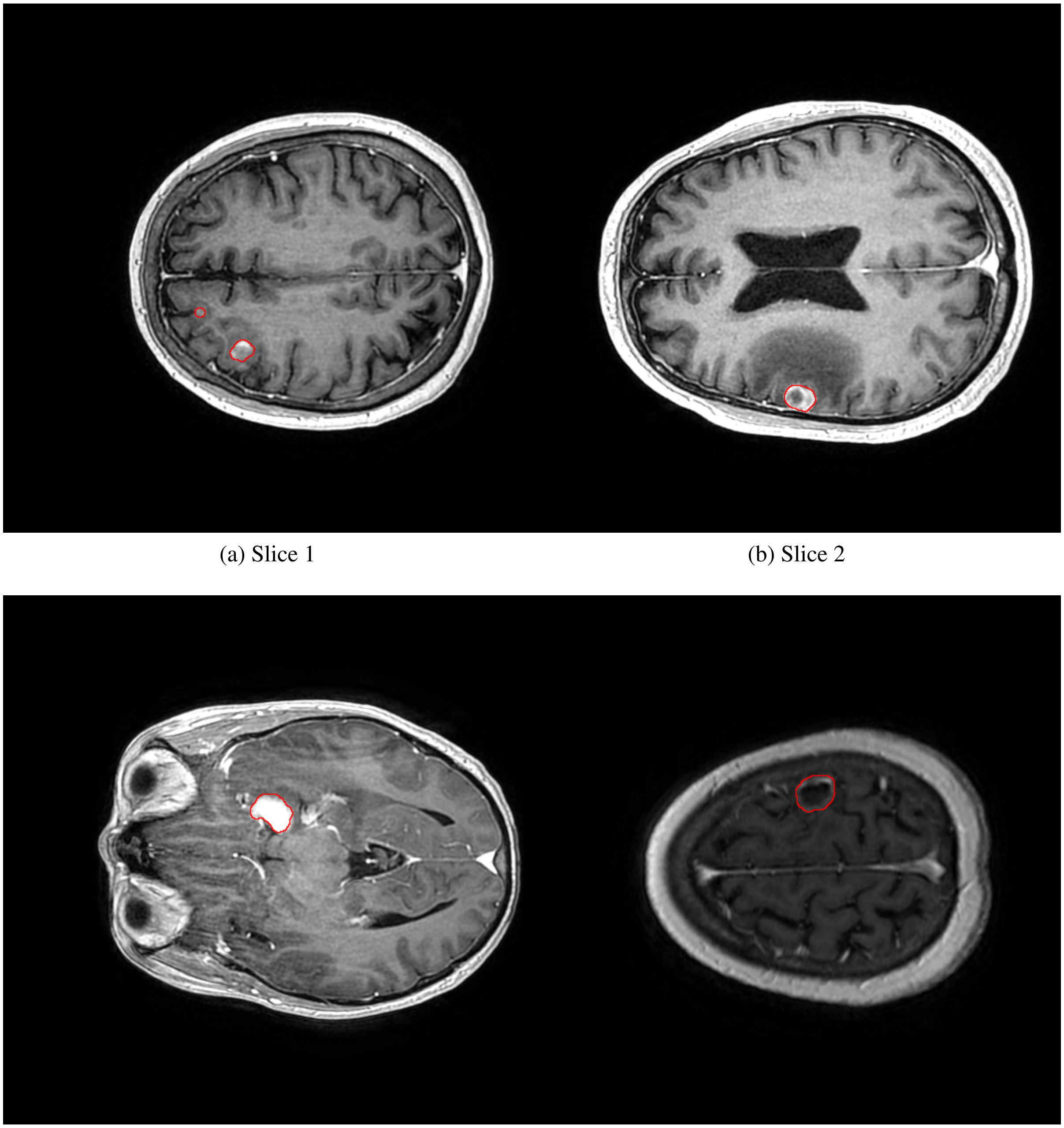
(a) Slice 1 (b) Slice 2

Figure 1.4: Volume is too dark. The left volume is too dark and the right volume looks incomplete. These two kinds of volumes are very different from other volumes.

The training set contains 30 volumes with 9098 crops. The validation set contains 8 volumes with 2350 crops. The test set contains 7 volumes with 148 slices. The training set is used for updating and determining the parameters of deep neural networks. The validation set is used for estimating how well the neural network has been trained and tuning some hyperparameters. If the whole data set is large enough, randomly selecting from them may be a good choice. But our data size is limited, so we should carefully determine how we divide the training, validation and test dataset and they are not randomly selected. The training set should be large enough because deep neural networks usually have lots of parameters to be learned and small size of training data will be very likely to cause overfitting. And tumor shape, location, size and appearance cases should be as diverse as possible in training data since to make the neural networks generalize better, we should enable neural networks to see lots of different training examples. The test set could be larger but, if the test set is larger, we will have less training and validation data, which will not be enough for neural networks to learn a good model. We adopt simple data augmentation only on training and validation data, i.e., horizontal and vertical flipping of the image crops. We add these augmented data into the whole training and validation data, tripling the number of training and validation images. No other data augmentation is used. Thus, by data augmentation, the training set contains 27294 image crops and the validation set contains 7050 image crops. The crop size is 128 × 128 pixels. The voxels are represented by 16-bit integers. The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS) challenge is organized in conjunction with the international conference on Medical Image Computing and Computer Assisted Interventions (MICCAI). Also, our dataset is different from BRATS dataset [42] and we only have tumor and healthy brain tissue, i.e., a binary classification problem. Annotations of BRATS dataset comprise the GD-enhancing tumor, the peritumoral edema, the necrotic and non-enhancing tumor, as described in [42].

1.2 Challenge for Brain Tumor Segmentation

The difficulties for brain tumor segmentation are mainly due to high variation of brain tumors in size, shape, regularity, location, and their heterogeneous appearance (e.g., contrast, intensity and texture variation for different tumors). Below we list some of these challenges as well as show some image examples from our dataset.



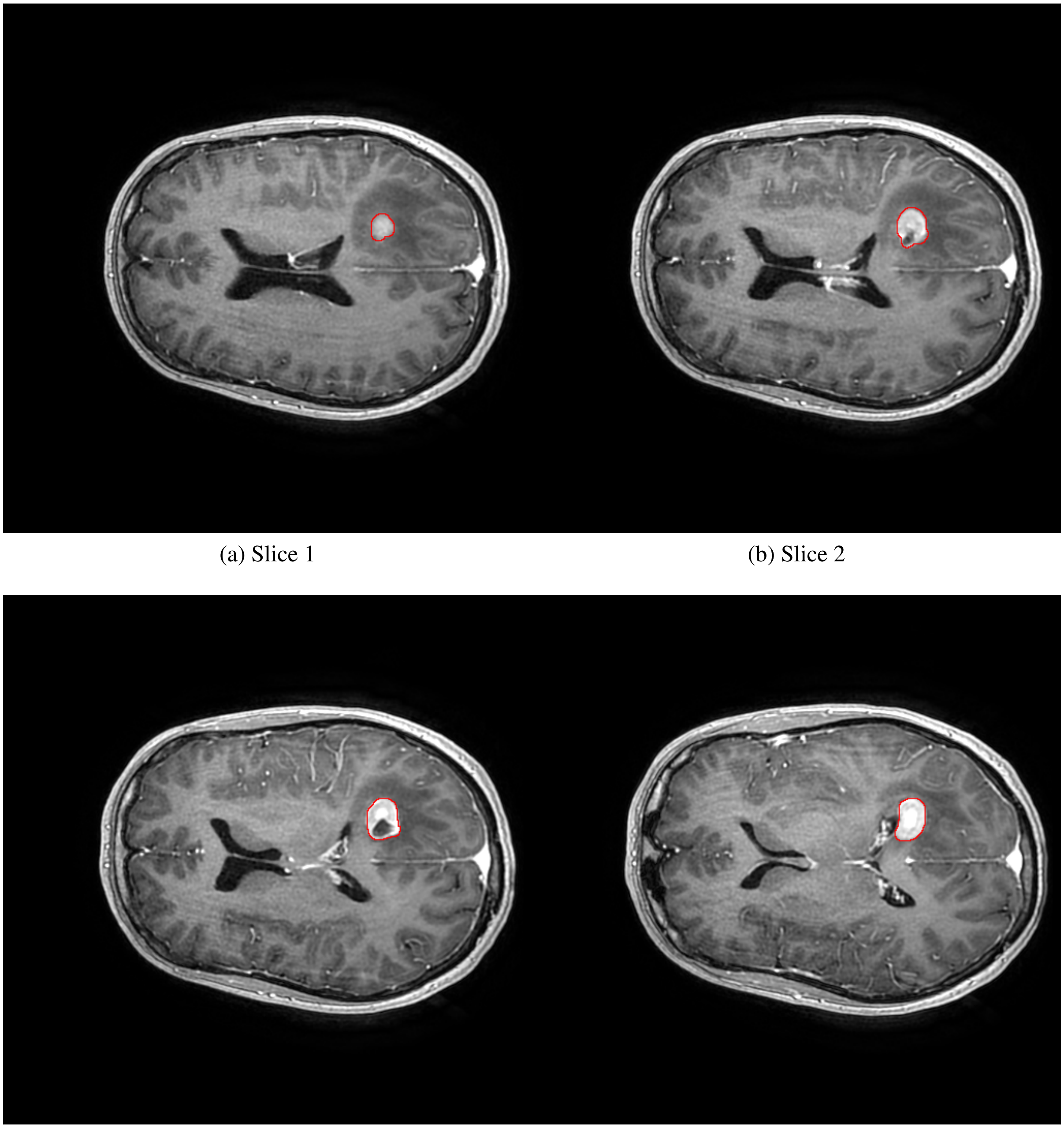
(a) Slice 1

(b) Slice 2

(c) Slice 3

(d) Slice 4

Figure 1.5: Examples of brain tumor diversity in size, location and shape. The red contours are the boundaries of the ground truth. The pixels inside the red contours are tumor pixels and the pixels outside the red contours are background pixels.



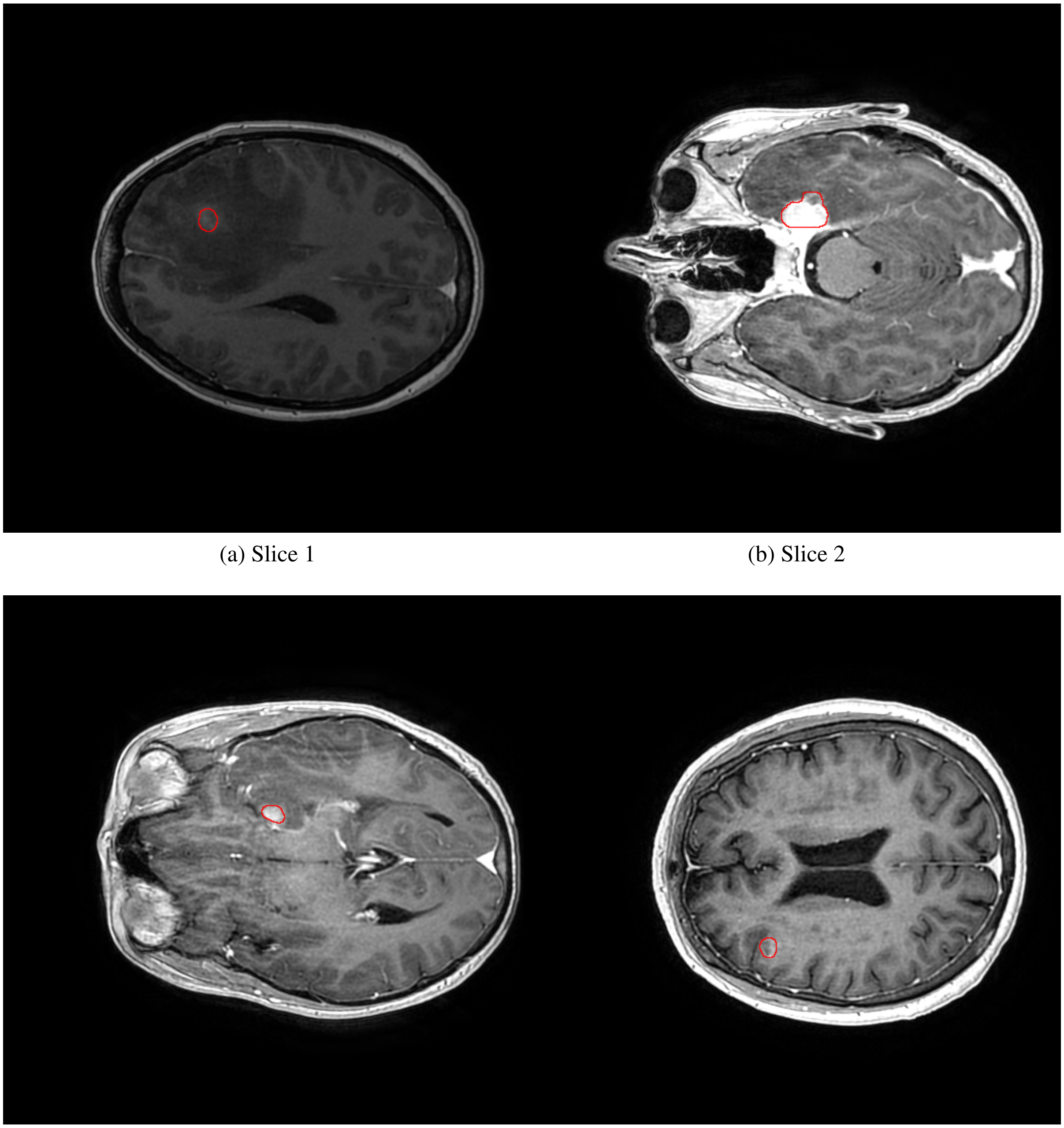
(a) Slice 1

(b) Slice 2

(c) Slice 3

(d) Slice 4

Figure 1.6: Examples of inconsistent appearance of tumor. The red contours are the boundaries of the ground truth. The pixels inside the red contour are tumor pixels and the pixels outside the red contours are background pixels.



(a) Slice 1

(b) Slice 2

(c) Slice 3

(d) Slice 4

Figure 1.7: Examples of similarity between tumor and healthy tissues. The red contours are the boundaries of the ground truth. The pixels inside the red contours are tumor pixels and the pixels outside the red contours are background pixels.

Diversity in Size, Location and Shape

The size and location of brain tumors are different for different patients. Also, for the same person, the size of a brain tumor varies from slice to slice. The variance of tumor size and location can be easily observed when going through the 3D MRI brain volume. As for tumor shape, we can observe its diversity in 2D slice images. The diversity of size, location and shape in brain tumors obstructs automatic segmentation to obtain satisfactory results. Figure 1.5 shows different size, location and shape of different tumors.

Highly Imbalanced Tumor over Background

Since the tumor area only accounts for a small area of the whole slice image in our dataset, usually less than 1%, segmenting tumors from background is a highly imbalanced dense prediction task, as indicated in Figure 1.1. When applying machine learning model to perform automatic segmentation, we should pay attention to dealing with this issue.

Inconsistency in Appearance

Different parts of one tumor may have different appearance even if they belong to the same connected component. For different slices of brain tumors, they usually have very different appearance. Figure 1.6 shows several examples with inconsistent tumor appearance. This adds to the difficulty of methods based on appearance models. Also, when applying machine learning approaches, the features extracted should capture these kinds of appearance variance.

Similarity between tumor and healthy tissue

Sometimes, surrounding healthy tissues are very similar to tumors. In this case, appearance is similar between the tumor and the background. If strong contrast boundaries exist between the tumor and healthy tissue, one can still segment the tumor by utilizing the intensity contrast information. Otherwise, it is difficult to locate tumor for an untrained human. In many cases, a tumor could be very similar to some other brain parts, as illustrated in Figure 1.7. This kind of tissues often further increase the difficulty of the segmentation process. When applying machine learning model, it will be hard to discriminate tumor from background and predict the labels correctly.

1.3 Overview of MRI-based Brain Tumor Segmentation

There are various existing interactive brain tumor segmentation approaches. An interactive brain tumor segmentation method using graph cuts is presented in [5]. 3D Slicer is used as an interactive tool for brain tumor segmentation [29]. A k-nearest neighbors algorithm based method [17] uses some minimum user interaction to segment a given brain by training and generalizing within that brain only. A Bayesian inference based interactive 3D brain segmentation is in [19], where user inputs are formulated as a probabilistic spatial term in a level set functional.

The previous automatic brain tumor segmentation approaches by using brain MRI scan are provided in [4]. The common procedure of current automatic or semi-automatice brain tumor segmentation methods include four phases, i.e., preprocessing of MRI, extracting features, applying machine learning models for tumor segmentation and postprocessing of segmentation results. In the preprocessing stage, certain image enhancement techniques, such as image denoising, intensity normalization and skullstripping, can be used for improving image quality. For enhanced images, commonly used image features, e.g., image intensity, local image texture, gradient features can be either extracted from one single modality or from multi-modal images. Various features from MRI can be used alone or combined together for automatic brain tumor segmentation.

Automatic segmentation algorithms can be categorized into classification and clustering methods according to whether they apply machine learning models in a supervised or an unsupervised way. First, these methods extract feature vectors, e.g., texture features, from voxelwise intensity. Then, the extracted feature vector is used as input to the machine learning model to decide for every voxel which class it belongs to. However, simple voxelwise classification or clustering methods do not make use of complete image information. Therefore, additional constraints, e.g., neighborhood regularization, shape and localization constraints are added into some methods. A random field regularization method is often imposed into neighborhood constraints, while deformable models can be used for shape constraints [4].

Clustering models work in an unsupervised way and cluster voxels into several groups based on certain similarity metrics [55]. On the other hand, other approaches for brain tumor segmentation employ classification models. Classification models require training data to learn a classification model using handcrafted features and ground truth annotations. Unlike clustering approaches, these approaches utilize little prior information on the brain structure, but they need to extract high dimensional low level image features. Then, using classification method directly models the relationship between these features and the label of a given voxel. These features could be raw input voxel intensity values [17] , local intensity histogram [41] and texture features such as Gabor filterbanks [51]. Classical discriminative learning techniques such as support vector machine [3] and decision and random forests [61, 14, 42] have also been applied to classify each voxel to be a certain tumor class.

Traditional classification machine learning models that segment tumor from background largely depeneds on hand-designed features. For these methods, the classifier is trained to separate healthy from non-healthy tissues assuming that the designed features have a sufficiently high discriminative power, i.e., they can differentiate tumor from background in general. One challenge with these methods based on hand-designed features is that they often need to compute a high dimension of features in order to accurately predict the class label for each voxel when working with many machine learning models. But, computing high dimensional features is expensive and requires large amounts of memory. More efficient techniques are employed by using lower dimensional features, e.g., utilizing dimensionality reduction or feature selection techniques, but the reduction in the dimension of features may decrease the prediction accuracy.

1.4 Convolutional Neural Network Approach for Brain Tumor Segmentation

Previous automatic brain tumor segmentation approaches depend on hand-engineered features that exploit very generic edge-related information for general computer vision tasks, but without specific adaptation to the domain of brain tumors. For automatic brain tumor segmentation, one would like to use features that are composed and refined into the domain of brain images. By utilizing these better features, the machine learning models are expected to be able to better segment the tumor from the background. Recently, deep convolutional neural networks (CNNs) have been successful in various computer vision tasks [49, 18, 36, 57, 10, 44, 13, 45] and can learn hierarchical features that are useful for various tasks. Preliminary investigations have demonstrated that applying deep CNNs for automatic brain tumor segmentation is a very promising approach [16, 62, 53].

Some previous methods first divide the 3D MR images into 2D or 3D patches. Then, a deep CNN is trained to predict the label of its center voxel. Urban et al. [53] and Zikic et al. [62] implemented a common deep CNN architecture, which consists of a series of convolutional layers and a non-linear activation function between each consecutive layer. And a final softmax output layer is used for calculating the probability of being each class. Havaei et al. [16] proposed a two-pathway CNN architecture that learns about the local brain structure information as well as larger context information. Obviously, these strategies [16, 62, 53] have two drawbacks. First, they are not efficient because the deep networks must be run independently for each patch, and there is a lot of redundancy when predicting the labels of nearby pixels due to overlapping patches between neighbouring pixels. Since these approaches are patch-based and once forward propagation can only predict one pixel label, they are very slow compared to dense prediction based approaches, which predict all pixel labels in one whole image by once forward computation [36, 57, 46]. Secondly, there is a trade-off between computational cost and the use of context. Larger patches require more computation, but can utilize more context information; while small patches allow the network to only see small context, but computation cost is low.

Current state-of-the-art approaches for semantic image segmentation [36, 57, 10, 44] all employ some form of fully convolutional networks, which can also be applied for brain tumor segmentation. Fully convolutional networks (FCNs) [36] take the image as input and output a probability map for each class and each pixel. Specifically, convolutional network architectures that had originally been developed for image classification can be successfully modified for dense prediction [36]. These modified networks significantly outperform the previous state-of-the-art approaches on challenging semantic segmentation benchmarks. CNNs for image classification integrate multi-scale contextual information via successive pooling and subsampling layers that reduce resolution to obtain a global prediction. In contrast, dense prediction for semantic image segmentation requires full-resolution output. Fully convolutional neural networks have been used for brain tumor segmentation [1, 25, 40, 47]. Dilated convolutional networks have been used for semantic segmentation by multi-scale context aggregation [57] and have been applied for brain tumor segmentation [34, 37, 60]. U-Net based convolutional network [46] has been used for biomedical image segmentation. Also, 3D U-Net [2, 8] is adopted for brain tumor segmentation.

1.5 Our Approach

The goal of this thesis is to evaluate the network architectures that have been successful in semantic image segmentation and brain segmentation to find the architecture that works best for our data. In addition, we use graph cut optimization to further improve the segmentation results by regularizing the segmentation boundary obtained from a neural network.

Since the tumor area only accounts for a small proportion of the whole brain slice in our dataset, usually less than 1%, segmenting tumors from background is a highly imbalanced dense prediction task. To deal with this issue, we crop the whole slice image into overlapping crops. Also, by doing this, we can speed up the training process compared to training on the whole slice. First, we train the deep convolutional neural network, including fully convolutional neural network [36], dilated residual network [57] and U-Net [46], to segment each crop of whole slices. Then, we combine the segmentation results of all crops together by a simple union operation when testing on test slice image. We consider performing the segmentation slice by slice from the axial view. Thus, the deep CNNs sequentially process each 2D slice.

Then, we apply graph cut optimization to further refine the neural network segmentation results. When we use energy minimization framework to perform segmentation, the two commonly used energy terms are the data term and smoothness term. Data term, also called regional term, contains regional appearance information. Smoothness term, or boundary term, encodes the coherence information among image pixels. Three deep convolutional neural networks adopted in this thesis take the image as input and output a probability map for each class and each pixel. Then, we convert these probabilities into different data terms and incorporate them into the graph cut energy minimization framework.

To reduce the shrinking effect of the graph cut algorithm, we introduce a ballooning bias as in [9]. Volume ballooning adds ballooning bias to the energy function so that the pixel can get a bonus if it is assigned to the foreground. Volume ballooning is encoded as an unary term in energy function and can be easily integrated into the energy optimization framework.

The main contributions we have are as following:

1. We introduce an approach that does not need user input as in [35, 9], and that performs

automatic brain tumor segmentation.

2. We evaluate previous successful fully convolutional networks to perform brain tumor segmentation.
3. We combine deep CNNs with graph cut regularization and use different data terms to improve the deep neural network segmentation results.

We optimize all the parameters of the graph cut segmentation by grid search on the validation data and use these parameters on test data.

1.6 Outline of This Thesis

This thesis is organized as follows: Chapter 2 is a review of the related works, which include the operations in convolutional neural networks used in this thesis and energy minimization framework. Chapter 3 introduces the details of the automatic brain tumor segmentation method. Chapter 4 presents experimental results. In Chapter 5, we conclude the thesis and discuss future work.

Chapter 2

Related Work

2.1 Deep Convolutional Neural Network

Deep learning [15] has been successful in computer vision, natural language processing and speech recognition. Convolutional networks [32], also known as convolutional neural networks (CNNs), are a special kind of neural network for processing data that has grid-like topology, e.g., image data, which can be thought of as a 2D grid of pixels. CNNs have achieved the state-of-the-art performance in various computer vision applications, including image recognition [49, 18], semantic segmentation [36, 57, 10, 44], object detection [13, 45], stereo matching [59], etc. Convolutional neural networks use a linear operation called convolution and are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

Convolutional networks were inspired by biological processes [22, 20, 21] in which the connectivity relationship between neurons is inspired by the anatomy structure of the animal visual cortex. Individual cortical neurons only respond to stimuli around a restricted area of visual field, also known as the receptive field. The receptive fields of different neurons partially overlap with each other such that these neurons cover the whole visual field. Neurophysiologists David Hubel and Torsten Wiesel collaborated for several years with neuroscientific experiments to explore and discover how the mammalian vision system works [22, 20, 21]. Their great discovery was that neurons in visual system responded very strongly to specific patterns of light, but they almost did not respond at all to other light patterns. These discoveries were long before the relevant computational convolutional network models were developed and have had great influence on contemporary deep learning models.

Convolutional neural networks are very similar to ordinary neural networks and are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product operation and optionally follows it with a non-linearity activation function. The entire neural network still represents a single differentiable function, i.e., from the raw input image pixels on one end to class scores at the other. Also, they still have a loss function (e.g. cross entropy) on the last layer. All the approaches that are developed for learning general neural networks parameters still apply to CNNs. CNN architectures make the explicit assumption that the inputs of the networks are images, which allows us to incorporate certain properties into the architecture.

Convolutional neural networks leverage three important ideas that can help improve a machine learning system, i.e., sparse interactions, parameter sharing and equivariant representations [15], which will make the forward function more efficient to be implemented and dramatically reduce the amount of parameters in the whole network. Also, convolution provides ways for dealing with inputs of variable size.

For traditional neural network layers, every output unit interacts with every input unit, as shown in Figure 2.1. However, convolutional neural networks typically have sparse interactions between input and output units, also referred to as sparse connectivity, which is achieved by setting the kernel size of CNNs smaller than the input size, as shown in Figure 2.2. For example, when processing an image, the input image might have thousands or millions of pixels, but we can still extract meaningful features, e.g., edges features, with kernels that only occupy a small and restricted area. There are several benefits by doing this. First, we only need to store fewer parameters, which reduces the memory requirements of the model. Figures 2.3 and 2.4 show the difference of the connectivity and parameters needed in convolutional network and fully connected network. They also show the receptive field difference of one unit in convolutional network and fully connected network. Second, it also means that computing the output usually requires much fewer numerical computation. These improvements in efficiency are often quite significant in real applications.

Parameter sharing refers to using the same parameter for more than one function in a neural network model, i.e., the value of the weight applied to one input is tied to the value of a weight applied elsewhere. In a traditional neural network, each element of the weight matrix is used exactly only once when computing the output of one layer. In a convolutional neural network, each member of the kernel is used at every pixel of the input image (except perhaps some pixels around the image boundary, depending on the design that uses some techniques about how to deal with the boundary pixels, e.g., zero padding around the input image boundary). The parameter sharing adopted by the convolution operation means that rather than learning a separate set of parameters for every location, we only need to learn one set of parameters that are shared by all locations. This will further reduce the memory storage requirements of the model. Thus, convolution operation is vastly more efficient than dense matrix multiplication in terms of memory requirements and the number of parameters to be learned.

In the case of convolution, the particular form of parameter sharing causes the convolution layer to maintain a property called equivariance to translation. For example, for 2D images, convolution layer creates a 2D map of where certain features appear in the input. If we move the object a bit in the input, the feature representation of the input will also move the same amount in the output.

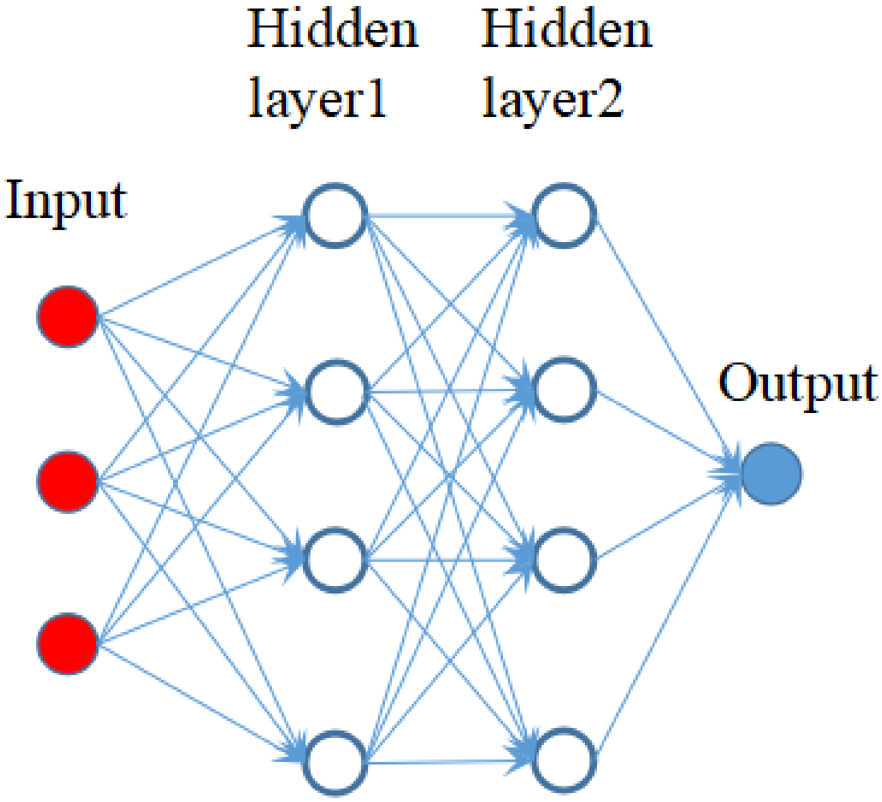


Figure 2.1: Examples of fully connected neural networks. The red, white and blue circles are the input, hidden and output units for the neural network. The arrows indicates the connection relationship between neuron units.

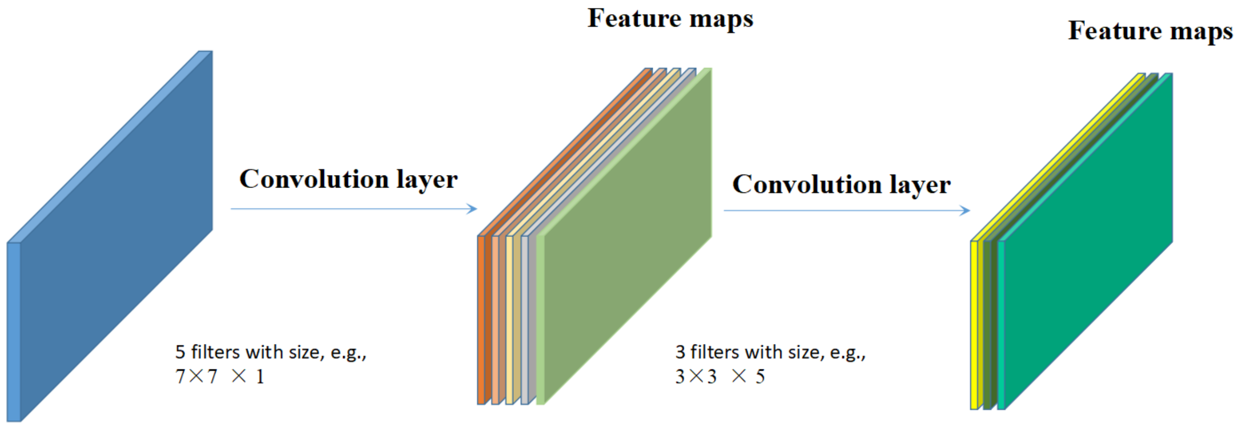


Figure 2.2: Examples of convolutional networks by applying a series of filters with size smaller than the input size. Each thin cuboid is one feature channel and is the convolution output of one filter applied on the previous input.

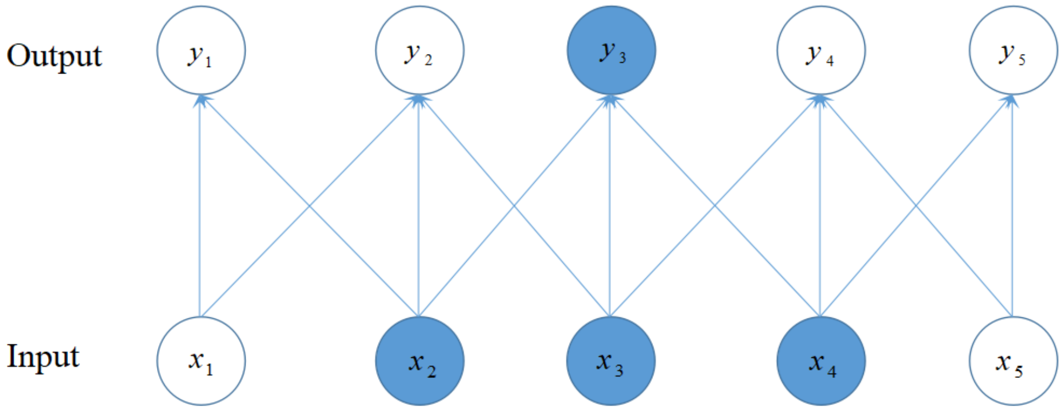


Figure 2.3: Examples of convolution operations. The above row is obtained by convolution with kernel size 3 applying to the bottom row. The arrows indicate which input units affect which output units. The blue circles in the bottom row affect the output y_3 and are called as the receptive field of y_3. Other units in the bottom row do not have influence on the output units. Each x_i is the input unit and y_i is the output unit. Image is from [15].

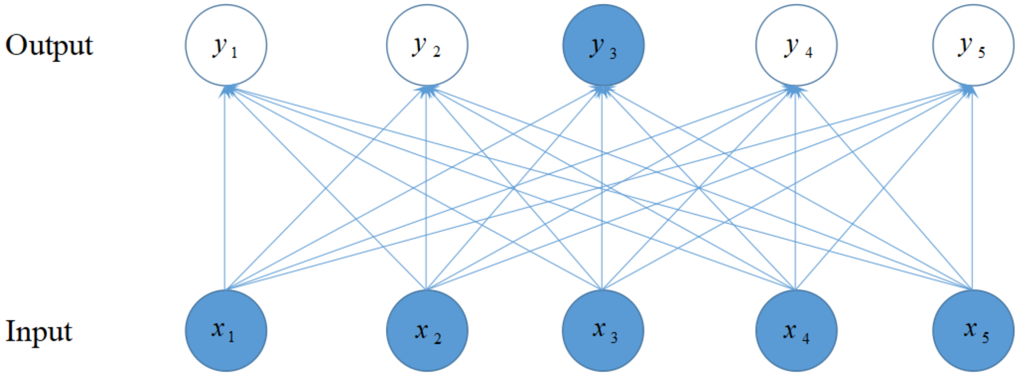


Figure 2.4: Examples of fully connected neural networks. The above row is formed by matrix multiplication with fully connectivity. The arrows indicate which input units affect which output units. All the units with blue circles in the bottom row affect the output y_3. Each x_i is the input unit and y_i is the output unit. Image is from [15].

In this chapter, we describe some operations in deep CNNs that are used in this thesis. We first describe what the convolution operation is. Next, we describe an operation that generalizes convolution operation called dilated convolution. Then, we describe an operation called pooling, which almost all convolutional networks employ. Then, we describe batch normalization, which is an useful technique to make it easier for training deep neural networks. Finally, we describe the challenges and optimization algorithms in training neural networks.

2.1.1 Convolution Operation

Usually, the operation used in a convolutional neural network does not correspond precisely to the definition of convolution as used in other fields such as engineering or pure mathematics. We describe several variants on the convolution function that are widely used in practice for neural networks.

In its most general form, convolution is an operation on two functions of a real valued argument. If we use a two-dimensional image I and a two-dimensional kernel K as the input, the convolution operation is:

$$(I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \tag{2.1}$$

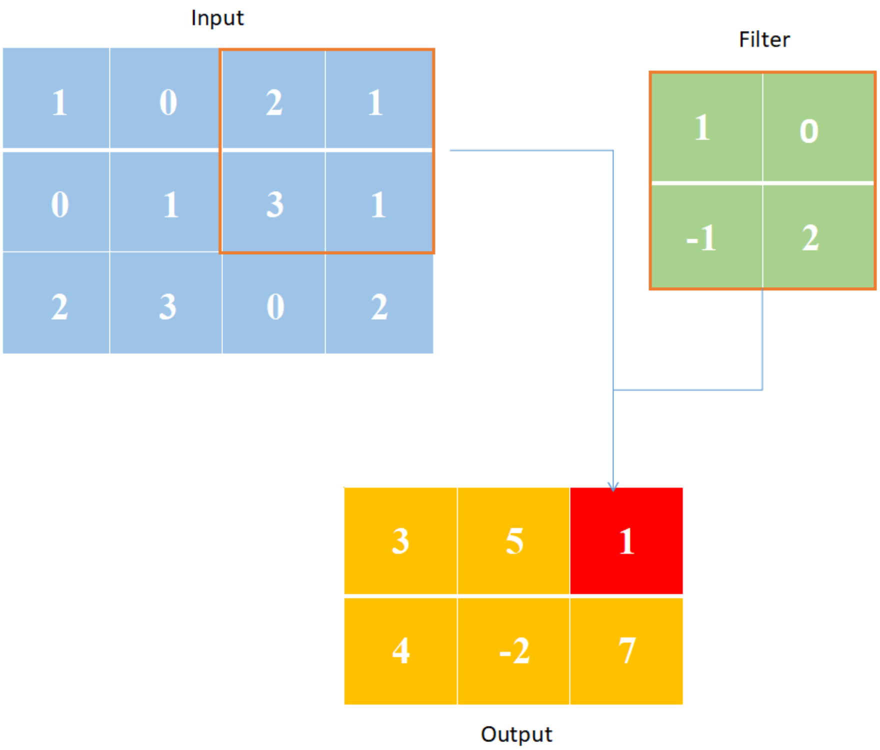
Convolution is commutative, meaning we can equivalently write:

$$(K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n) \tag{2.2}$$

Usually the Equation 2.2 is more straightforward to implement in a deep learning library, because there is less variation in the range of valid values of m and n. The commutative property of convolution arises because we have flipped the kernel relative to the input, in the sense that as m increases, the index into the input increases, but the index into the kernel decreases. The only reason to flip the kernel is to obtain the commutative property. Many deep learning libraries implement a related function called the cross-correlation, which is the same as convolution but without flipping the kernel. Figure 2.5 shows an example of convolution operation.

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n) \tag{2.3}$$

Equation 2.3 defines the cross-correlation operation.



1	0	2	1
0	1	3	1
2	3	0	2

1	0
-1	2

3	5	1
4	-2	7

Figure 2.5: An example of 2-D convolution with a 2 by 2 filter without kernel-flipping to the input and obtain output. The red square output is the dot product between the square input and the filter.

2.1.2 Dilated Convolution

Let $I : Z^2 \to R$ be a discrete function. Let $\Omega_r = [-r, r]^2 \cap Z^2$ and $K : \Omega_r \to R$ be a discrete filter of size $(2r + 1)^2$. The discrete convolution operator $*$ on input I and filter K can be defined as the following:

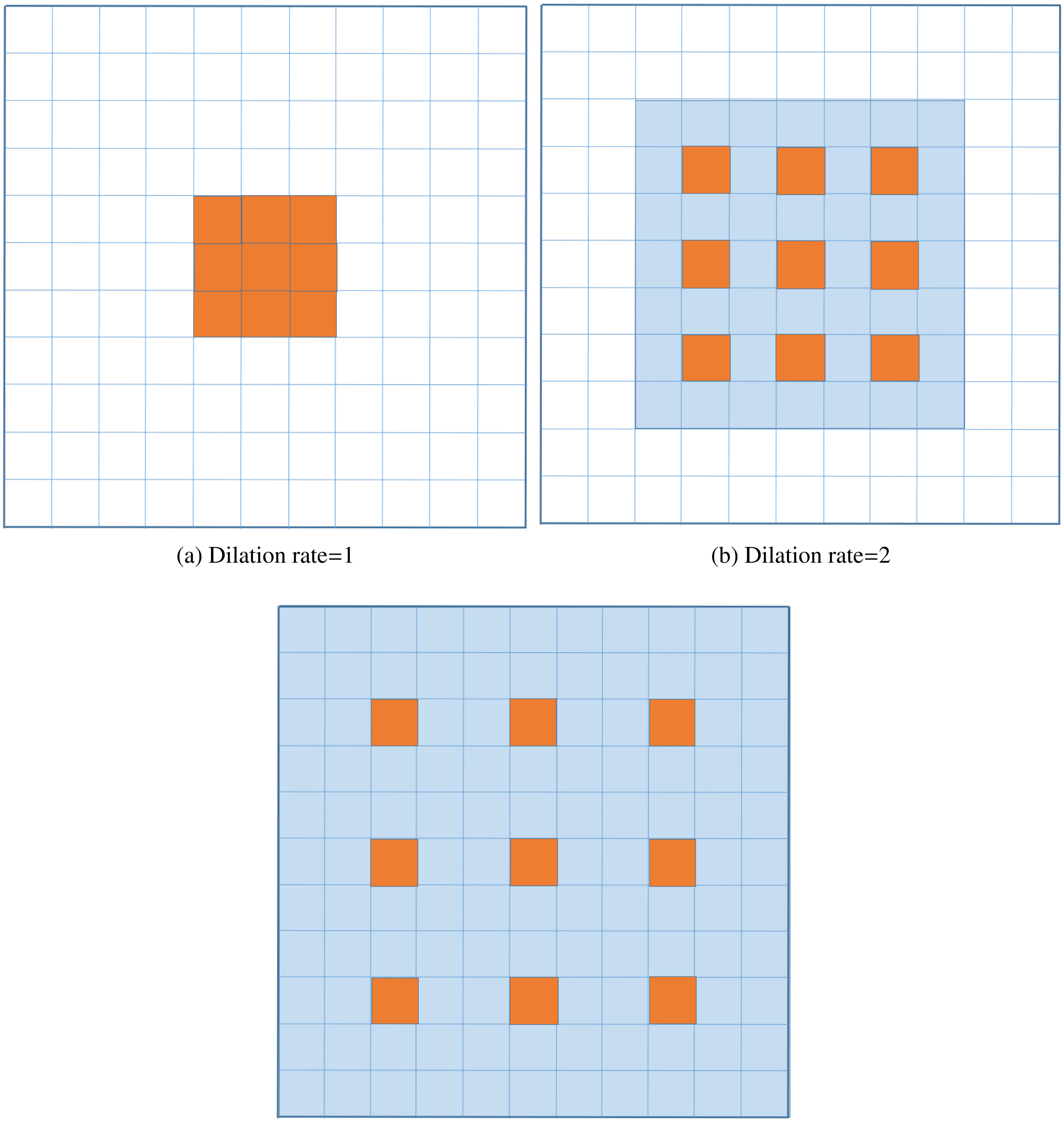
$$(I * K)(\mathbf{p}) = \sum_{\mathbf{s}+\mathbf{t}=\mathbf{p}} I(\mathbf{s})K(\mathbf{t}) \tag{2.4}$$

Let l be a dilation factor which determines the amount of spaces between neighbouring elements of the convolution operation and let operator $*_l$ be defined as the following:

$$(I *_l K)(\mathbf{p}) = \sum_{\mathbf{s}+l\mathbf{t}=p} I(\mathbf{s})K(\mathbf{t}) \tag{2.5}$$

The operator $*_l$ is a generalization of the convolution operator $*$. Dilated convolution is the convolution operation with filters that have spaces between each element, called dilation. Then, it can be used for increasing the receptive field but without increasing the number of parameters. Figure 2.6 shows three examples of the dilated convolution operation. Operator $*_l$ is called a dilated convolution or an l-dilated convolution [57]. The familiar discrete convolution $*$ is simply the 1-dilated convolution. The dilated convolution operator has been referred to in the past as convolution with a dilated filter. It plays a key role in the algorithme *à* trous, an algorithm for wavelet decomposition [38, 48].

Dilated convolution can be used for aggregating multiscale contextual information and support exponential expansion of the receptive field without losing resolution [57]. Thus, dilated convolution operator is well suited to dense prediction tasks. Semantic image segmentation is a dense prediction task, i.e., compute a label for each pixel in the image and requires pixel-level accuracy with multi-scale contextual information. Dilated convolution can be used for simplifying the deep network structure for semantic segmentation [57].



(a) Dilation rate=1

(b) Dilation rate=2

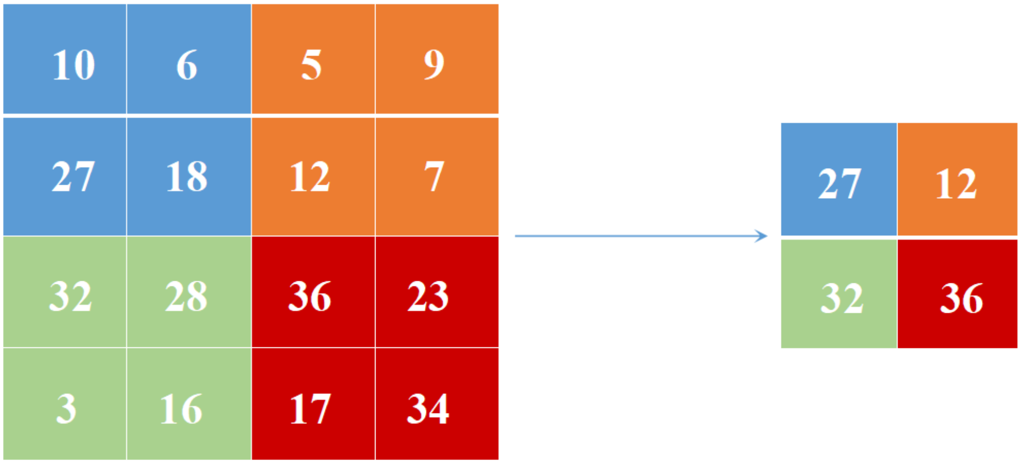
(c) Dilation rate=3

Figure 2.6: Examples of convolutions with different dilation rates. The orange squares are the locations where the convolution operates and the blue squares are the locations which are filled with zeros and have no effects on the output. The number of parameters associated with each layer is identical. The receptive field of output units are increased by using larger dilation factors.

2.1.3 Pooling

A typical convolutional neural network layer consists of three steps. First, the convolutional layers perform several convolutions to produce a set of linear activations. Second, a nonlinear activation function, e.g., the rectified linear activation function and softmax activation function, is applied for each linear activation. In the third step, a pooling function will be used to modify the nonlinear activations further.

A pooling operation computes a summary statistic of the nearby outputs of the previous layer at a certain location. For example, the max pooling operation takes the maximum value within a rectangular neighborhood, as shown in Figure 2.7. Other pooling functions include the average of a rectangular neighborhood, the L_2 norm of a rectangular neighborhood, etc. The intuition behind using pooling is that the precise location of a feature is less important than its rough location. The pooling layer can reduce the resolution of the feature representations, the number of parameters and amount of computation. In some sense, it is useful for controlling overfitting. It is common to apply a pooling layer between successive convolutional layers in deep CNNs. A pooling operation makes the feature representations become approximately invariant to small translation. Invariance to translation indicates that if the input is translated by a small amount, most of the pooled values do not change. Invariance to local translation will be useful when some feature is present is more important the exact location.



10	6	5	9
27	18	12	7
32	28	36	23
3	16	17	34

27	12
32	36

Figure 2.7: Applying 2 by 2 max pooling to the left activation map and obtain the right output.

2.1.4 Batch Normalization

Batch normalization [24] is an adaptive reparametrization method and is used for solving the difficulty of training very deep models. In practice, it is very useful for making it easier to optimize deep neural networks.

Essentially, very deep models are the composition of many functions. Training very deep networks is complicated because the inputs to each layer are affected by the parameters of all preceding layers, and the parameters of all the layers are updated simultaneously. When we update the parameters, unexpected results can happen, e.g. gradient vanishing and gradient explosion, because many functions or layers composed together are updated simultaneously. In these cases, it will be very difficult to choose an appropriate learning rate, e.g., for stochastic gradient descent, because the effects of updating the parameters of one layer strongly depend on the parameters of all the other layers.

Batch normalization provides a simple and effective way of reparametrizing almost any deep network. The reparametrization of a deep neural network can significantly simplify the problem of coordinating updates across many layers. Batch normalization can be used for any input or hidden layers in a neural network. In the following, we describe how batch normalization works.

Consider a mini-batch $\mathcal{B}$ of size m. Since the normalization is applied to each activation independently, let us focus on a particular activation x. Thus, we have m values of this activation in the mini-batch, which are denoted by $\mathcal{B} = \{x_{1...m}\}$. The procedure of batch normalization is shown in the following steps.

First, normalize the batch of values $\mathcal{B} = \{x_{1...m}\}$ using the mean and variance of the batch calculated as:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.6}$$

and

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2, \tag{2.7}$$

where $\mu_{\mathcal{B}}$ is the mini-batch mean and $\sigma_{\mathcal{B}}^2$ is the mini-batch variance.

Then, calculate the normalized values $\hat{x}_{1...m}$ as:

$$\hat{x}_{1...m} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, \tag{2.8}$$

where ϵ is a small positive constant imposed to avoid encountering undefined denominator by normalization. Then, we apply linear transformations to the normalized values $\hat{x}_{1...m}$:

$$y_i = \gamma \hat{x}_i + \beta. \tag{2.9}$$

The variables γ and β are learned parameters that allow the new normalized values $y_{1...m}$ to have any mean and standard deviation. Normalizing the value of a unit can reduce the expressive power of the neural network containing that unit. In order to maintain the expressive power of the network, it is common to replace the batch of hidden unit activations $x_{1...m}$ with $y_{1...m}$. The new parametrization can represent the same family of functions of the input as the old parametrization of the network, but the new parametrization of the network is much easier to be trained with gradient descent.

In convolutional networks, we need to apply the same normalizing μ and σ at every location in a feature map to ensure that the statistics of the feature map are the same across all the spatial locations [15].

2.1.5 Training the neural networks

Optimization algorithms are important in training deep neural networks because the training process is expensive. Researchers have developed a specialized set of optimization algorithms for solving it.

Training neural networks involves solving one particular case of the optimization problems: finding the parameters $\mathbf{W}$ of a neural network that minimize a loss function $\mathcal{L}(\mathbf{W})$ which typically consists of a performance measure evaluated on the training data and additional regularization terms. Optimization algorithms for training deep neural networks also typically include some special structure of machine learning optimization objective functions.

Difficulties of training deep neural networks

The difficulty of general optimization can be avoided in traditional machine learning model if we carefully design the objective function and constraints so that the optimization problem is convex. However, we must deal with the general non-convex optimization problem when training neural networks. There are several challenges, e.g., ill-conditioning of the Hessian matrix, local minima, saddle points, etc [15], when optimizing the loss function of neural networks.

(1) There are still some challenges even when optimizing convex functions. The most difficult issue in general optimizing problems is the ill-conditioning of the Hessian matrix. The ill-conditioning problem is generally believed to be present when training neural networks

and can cause stochastic gradient descent to increase the loss value even with a very small learning rate.

(2) For a convex optimization problem, any local minimum is also a global minimum. Non-convex functions, e.g., the loss function of deep neural networks, can have many local minima. In practice, there will be an exponentially large number of local minima for the loss function of very deep neural networks. Local minima can be problematic if they have high loss values in comparison to the global minimum. If the local minima with high cost are common, the optimization problem will be difficult. But for sufficiently large neural networks, most local minima have low loss function values, and it is often acceptable to find a set of parameters with a low loss function value, but not necessarily global minimum. In this case, local minima are not major problems.

(3) For many high dimensional non-convex functions, local minima (maxima) are rare and saddle points are more common. In many real applications, the number of saddle points is far more than the number of local minima. A saddle point of a function is a stationary point, where its derivative or gradient is zero, but not an extremum. Some points around a saddle point have larger function values than the saddle point, while other points have smaller function values. At a saddle point, the Hessian matrix has both positive and negative eigenvalues. The gradient magnitude around a saddle point is often small. On the other hand, first order methods, including gradient descent, block coordinate descent, etc, almost always escape saddle points [33], but gradient descent may take exponential time to escape saddle points [12].

(4) There may also be wide and flat regions with constant value. In these regions, the gradient as well as the Hessian matrix are all zero. These degenerate cases are major problems for numerical optimization algorithms. For a convex optimization problem, a wide and flat region must consist of all the global minima, but for a general optimization problem, such a region may have a high objective function value.

Optimization algorithms for training neural networks

In this section, we describe some commonly used optimization algorithms, e.g., stochastic gradient descent, Nesterov' s accelerated gradient [43] and Adam [30], for training deep learning models, as described in [26]. Assume the goal is to minimize the loss function. For a dataset D, the optimization objective function is the average loss over all $|D|$ data instances, as shown in the following:

$$\mathcal{L}(W) = \frac{1}{|D|} \sum_{i}^{|D|} f_W(X^i) + \lambda r(W), \tag{2.10}$$

where $f_W(X^i)$ is the loss evaluated on data instance X^i and $r(W)$ is the regularization term with weight λ. In practice, the data size $|D|$ could be very large, so a stochastic approximation of the objective function, i.e., by drawing a mini-batch with size N from the whole $|D|$ data instances, is adopted:

$$\mathcal{L}(W) \approx \frac{1}{N}\sum_{i}^{N} f_W(X^i) + \lambda r(W) \tag{2.11}$$

The model computes $f_W(X^i)$ in the forward propagation and the gradient ∇f_W in back propagation. The parameters update $\triangle W$ consists of the gradient ∇f_W and the regularization term gradient ∇r_W. In the following section, we describe three commonly used optimization algorithms for training deep neural networks.

(1) **Stochastic gradient descent**

Stochastic gradient descent (SGD) uses a linear combination of the previous weight update V_t and the negative gradient $\nabla\mathcal{L}(W)$ to update the weights W. The learning rate α is the weight on the negative gradient $\nabla\mathcal{L}(W)$. The momentum μ is the weight on the previous update V_t. SGD computes the updated values V_{t+1} and the updated weights W_{t+1} at iteration $t+1$ using current weights W_t and the previous weight update V_t by the following way:

$$V_{t+1} = \mu V_t - \alpha\nabla\mathcal{L}(W_t) \tag{2.12}$$

and

$$W_{t+1} = W_t + V_{t+1}, \tag{2.13}$$

where α and μ are the learning hyperparameters.

(2) **Nesterov' s accelerated gradient**

Nesterov' s accelerated gradient (NAG) appears in [43] and achieves a convergence rate of $O(1/t^2)$ instead of $O(1/t)$ for convex optimization problems. Because of the non-smoothness and non-convexity of the loss function of the deep neural networks, the assumptions in the NAG algorithms [43] do not hold. In practice, NAG is a very effective method when optimizing certain types of deep neural network architectures [52].

The weight update formulas are:

$$V_{t+1} = \mu V_t - \alpha\nabla\mathcal{L}(W_t + \mu V_t) \tag{2.14}$$

and

$$W_{t+1} = W_t + V_{t+1}. \tag{2.15}$$

(3) **Adam**

The Adam [30] is a gradient-based optimization method and includes an adaptive moment estimation (m_t, v_t). The update formulas are shown as:

$$(m_t)_i = \beta_1 (m_{t-1})_i + (1 - \beta_1)(\nabla \mathcal{L}(W_t))_i \tag{2.16}$$

and

$$(v_t)_i = \beta_2 (v_{t-1})_i + (1 - \beta_2)(\nabla \mathcal{L}(W_t))_i^2. \tag{2.17}$$

Kingma et al. [30] proposed to use $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$ as default parameter values.

2.2 Energy Minimization Framework

In this section, we describe the energy minimization framework and three energy function terms, including data term, smoothness term and volumetric bias term.

2.2.1 Energy Minimization

Image segmentation can be formulated as a labeling problem, where the problem is to assign each pixel $p \in \Omega$, a label f_p from some finite set L. The goal is to find a labeling f that assigns each pixel $p \in P$ a label $f_p \in L$, where f is both piecewise smooth and consistent with the observed data. Let $\boldsymbol{f}$=$[f_p \in L | \forall p \in \Omega]$ be the labeling of all pixels. This problem can be expressed naturally in terms of energy minimization [7]. One formulates the energy function $E(\boldsymbol{f})$ that typically consists of unary term $D_p(f_p)$ and pairwise term $V_{pq}(f_p, f_q)$. A standard energy function can be formulated as:

$$E(\boldsymbol{f}) = \sum_{p \in \Omega} D_p(f_p) + \lambda \sum_{pq \in \mathcal{N}} V_{pq}(f_p, f_q). \tag{2.18}$$

In Equation (2.18), $D_p(f_p)$ is called data term, which is the cost when pixel p is assigned to label f_p. $V_{pq}(f_p, f_q)$ is the smoothness term, which regularizes segmentation discontinuities. λ adjusts the relative importance between the data term and the smoothness term.

2.2.2 Data Term

As discussed in Section 2.2.1, the data term measures how well the labeling fit into the observed data and is usually described by the appearance model. In the binary segmentation case, it represents the cost for assigning a pixel p to be object or background. The negative log-likelihood of the image intensity histogram is often used as data term and is also adopted in [35, 9]. The log-likelihood function, $logPr$, means the log probability of the pixel p assigned to the corresponding label. Higher probability means that the pixels are more likely to be assigned to the corresponding label, and thus incurs a low energy cost. The minus sign in front of $logPr$ just implies small cost for high $logPr$ and large cost for low $logPr$. The data terms are calculated as:

$$D_p(BKG) = -logPr(I_p|BKG) \tag{2.19}$$

and

$$D_p(OBJ) = -logPr(I_p|OBJ), \tag{2.20}$$

where I_p denotes the image intensity of pixel p. OBJ is the object label while BKG represents the background label in our binary label set $L = \{OBJ, BKG\}$. Figure 2.8 shows how to construct part of the graph corresponding to the data term.

2.2.3 Smoothness Term

The smoothness term in the energy function is a pairwise term encouraging the coherence between neighboring pixels and can be thought of as the cost for the segmentation discontinuity, i.e., different labels between neighbouring pixels. Hence, these penalties are on the boundary of the segmentation. Since our goal is to minimize the energy function, this smoothness term encourages the segmentation results to have a shorter boundary. The smoothness term we use in this thesis is shown as below:

$$V_{pq}(f_p, f_q) = w_{pq} \cdot \delta(f_p, f_q), \tag{2.21}$$

where

$$\delta(f_p, f_q) = \begin{cases} 1 & f_p \neq f_q, \\ 0 & otherwise. \end{cases}$$

and

$$w_{pq} = exp(-\frac{(x_p - x_q)^2}{2\sigma^2})\frac{1}{dist(p,q)}, \tag{2.22}$$

where $dist(p,q)$ is the distance between the pixel p and pixel q.

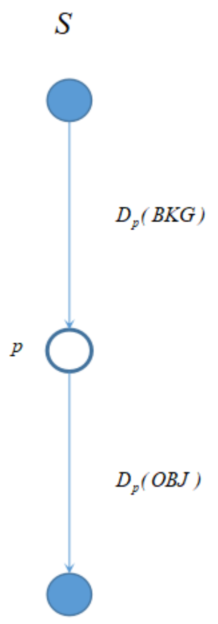


Figure 2.8: T-links weights for a pixel p in the graph

Each brain volume has different sampling rate along dimensions x, y and dimension z. $dist(p,q)$ for two pixels on the same slice is different from those on different slices. We adopt 6-neighborhood system in this thesis, as indicated in Figure 2.9.

2.2.4 Optimization with Graph Cuts

Graph cuts have various applications in computer vision problems, e.g., image segmentation, image restoration, stereo correspondence and other ones which can be formulated in terms of energy minimization [54]. It can also be applied to other real problems which are beyond computer vision, e.g., airline scheduling and bipartite matching. Max flow and min cut are the cornerstone problems in combinatorial optimization.

A global minimum solution can be guaranteed by graph cuts for certain energy functions. Also, the energy function optimized by graph cuts can be easily extended to an ND setting.

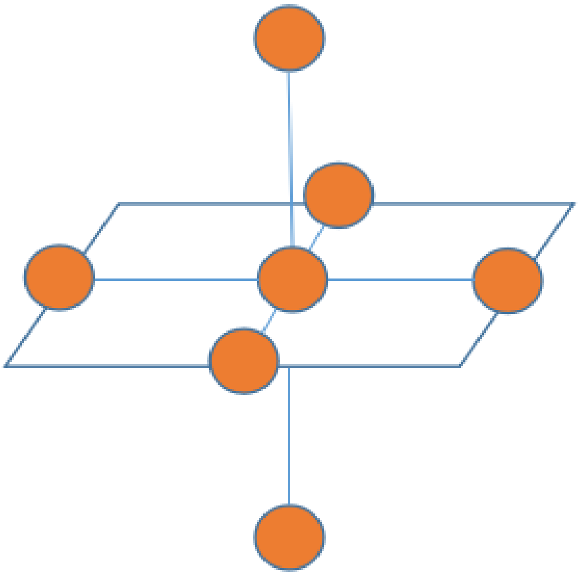


Figure 2.9: 6-neighborhood system. Each voxel has four connections to its immediate left, right, top, bottom neighboring pixels within the same slice, and two connections to its closest neighbor pixels in the previous and next slice.

Hence, graph cuts are adopted as the basic segmentation algorithm after obtaining the segmentation results by neural networks in this thesis.

Let $G = \{V, E\}$ be a weighted graph where V is a set of vertices and E is a set of weighted edges connecting the nodes in V. There are two special terminal vertices, the source vertice s and sink vertice t. An $s - t$ cut, $C = (S, T)$, is that if removing a subset of edges C from E, all vertices are partitioned into two disjoint sets, $s \in S$ and $t \in T$ and there is no path from s to t. The minimum $s - t$ cut problem is to find a cut C with the minimum cost, which is the total weight of the removed edges.

Figure 2.10 shows a simple 3×3 graph with two terminal vertices s and t, and its minimum $s - t$ cut. The thickness of the edges corresponds to the magnitude of the edge weights, i.e., thick edge represents large weight while thin edge represents small weight. The green dashed curve illustrates the minimal s-t cut on this graph. According to the theorem of Ford and Fulkerson [27], finding the min-cut on the graph is equivalent to the max-flow problem. From Figure 2.10, we can understand the relationship between finding the min-cut on the graph and applying the energy minimization to find the labeling for each pixel. If the weight of each edge corresponds to the energy term, a cut with minimum cost corresponds to the labeling for all pixels with the minimum energy function value.

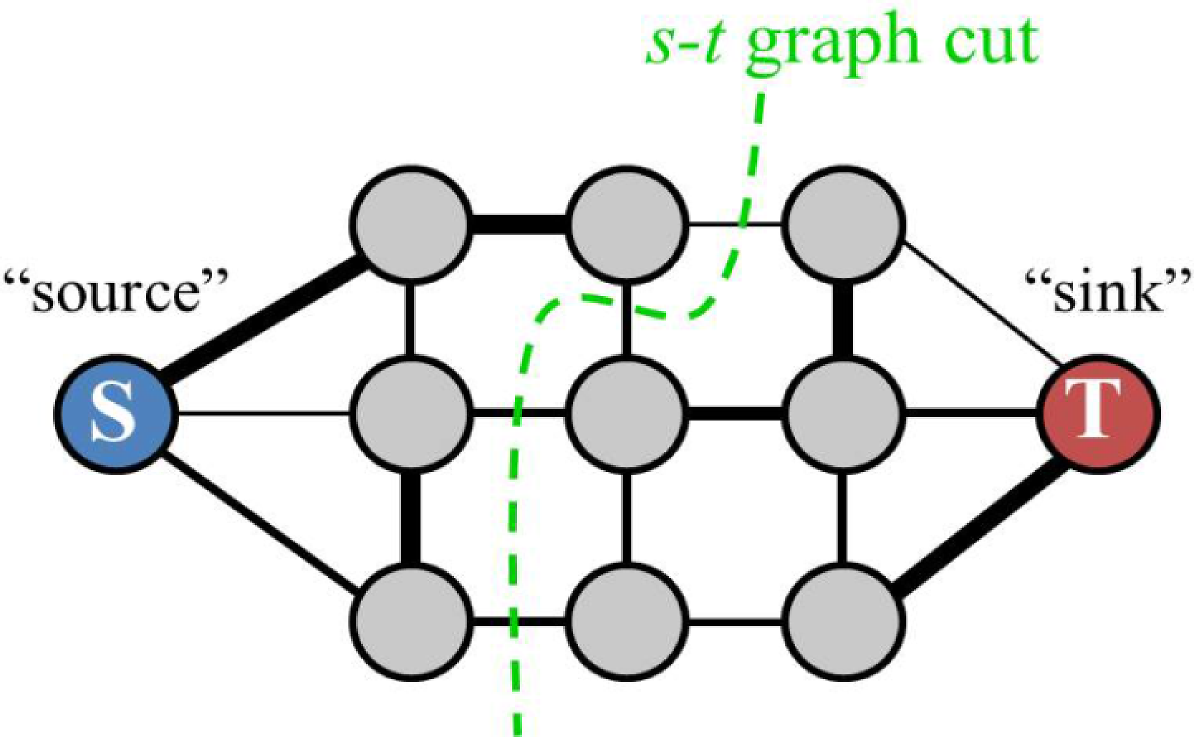


Figure 2.10: An $s - t$ cut on graph with two terminals. [Image credit: Yuri Boykov]

2.2.5 Binary Segmentation with Graph Cut

This thesis aims to segment tumor tissues from other healthy tissues. Since there are only two classes, i.e., the tumor and healthy tissues, this is a binary segmentation task. Each pixel is assigned a label from a label set $L = \{0, 1\}$, where 1 represents the label of the tumor and 0 corresponds to the label of the background. The energy function for binary segmentation is:

$$E(\boldsymbol{f}) = \sum_{p \in \Omega} D_p(f_p) + \lambda \sum_{pq \in \mathcal{N}} V_{pq}(f_p, f_q), \tag{2.23}$$

where Ω is the set of all pixels in the image and $\mathcal{N}$ is the neighborhood system on Ω, and pixel pairs (p, q) are neighbouring pixels in the adopted neighborhood system. $f_p \in L$ denotes the label assigned to pixel p. $V_{pq}(f_p, f_q)$ is the penalty for neighbouring pixels when they are assigned to different labels:

$$V_{pq}(f_p, f_q) = w_{pq} \cdot \delta(f_p, f_q), \tag{2.24}$$

where

$$\delta(f_p, f_q) = \begin{cases} 1 & f_p \neq f_q, \\ 0 & otherwise. \end{cases}$$

For $V_{pq}(f_p, f_q)$, when neighbouring pixels have the same labels, the penalty will be zero; otherwise there will be penalty for assigning different labels. This is indicated by:

$$V_{pq}(0, 0) = 0 \tag{2.25}$$

and

$$V_{pq}(1, 1) = 0 \tag{2.26}$$

Figure 2.11 shows a simple example of 3×3 image for binary segmentation. First, we construct a graph based on the original image using unary term and smoothness term. Each pixel corresponds to a node in the graph. There are two special terminal nodes, the source node s and the sink node t, which represent the set of labels that can be assigned to each pixel. The thickness of the edges corresponds to the magnitude of the edge weights, i.e., thick edge represents large weight while thin edge represents small weight. There are two types of edges in the graph. One is between neighboring pixels, usually called $n - links$ whose weights reflect the penalty for discontinuity between neighboring pixels and are derived from the pair-wise smoothness term in the energy function. The other type of edges connect the pixel node and the terminal node, often called $t - links$ whose weights represent the cost for being the corresponding label and are determined by the data term in the energy function.

Graph cuts can be used for optimizing the energy function that corresponds to the graph built previously. A minimum cut, labeling the pixels in the image into two disjoint sets, can be obtained. The two disjoint sets represent the binary segmentation, as shown in Figure 2.11.

Not all kinds of energy functions can be optimized by graph cuts, Kolmogorov [31] showed that only submodular energy functions can be optimized by graph cuts. In binary labeling case, the energy function needs to satisfy the following inequality:

$$V_{pq}(0, 0) + V_{pq}(1, 1) \leq V_{pq}(1, 0) + V_{pq}(0, 1), \tag{2.27}$$

where, $\{0, 1\}$ is the label set and V_{pq} is the pair-wise smoothness term in the energy function involving the neighbouring pixels p and q. Otherwise, minimizing the energy function is a NP hard problem. It can be verified from Equation (2.21) that the energy function for binary segmentation satisfies the submodular condition. Therefore, the energy function can be optimized by graph cuts.

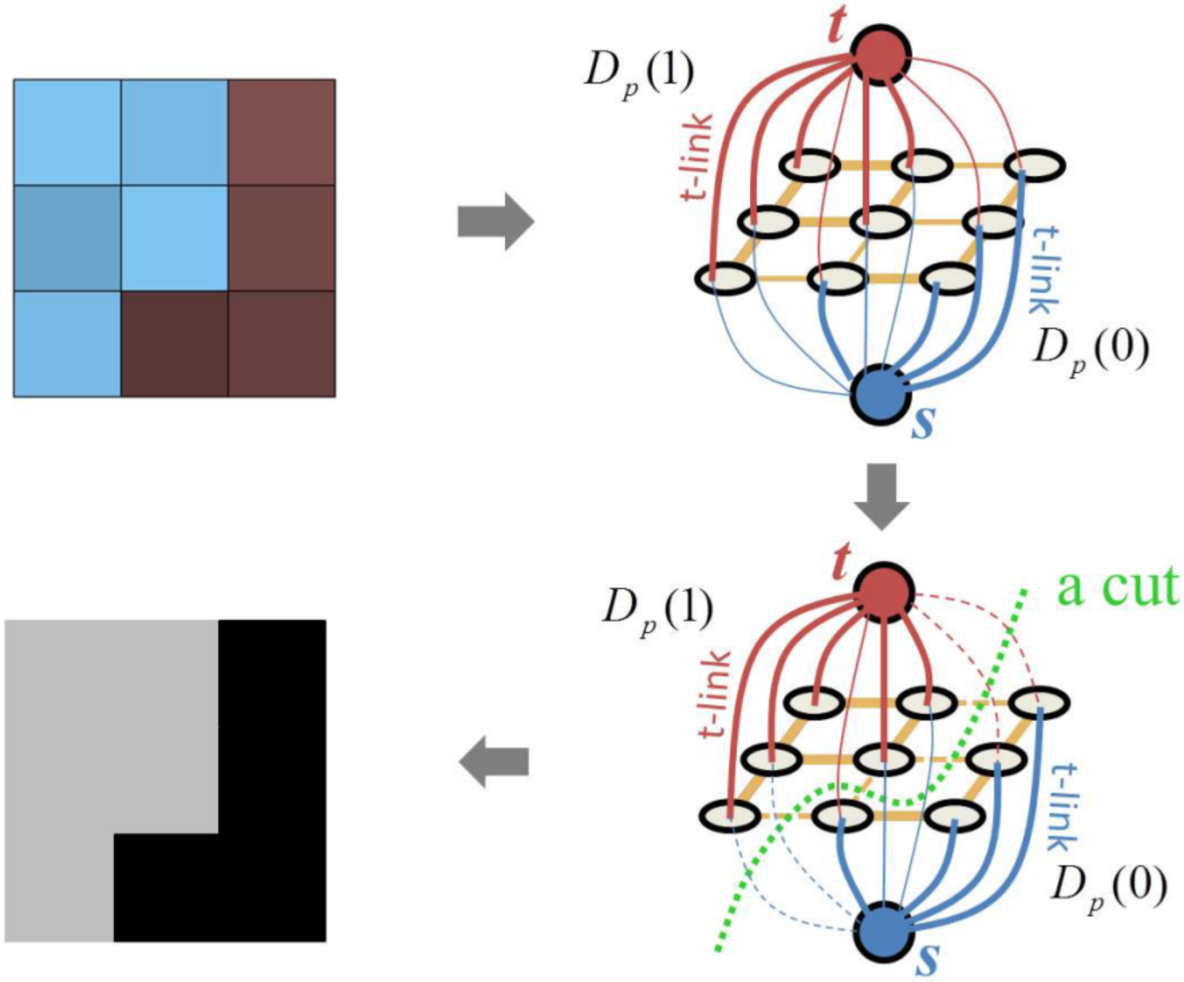


Figure 2.11: Binary segmentation for 3×3 image. (top-left): Original image; (top-right): Graph constructed based on original image with extra s and t terminals; (bottom-right) A minimal cut for the graph separating all pixels into two disjoint sets; (bottom left): Segmentation result, one color stands for one label. [Image credit: Yuri Boykov]

2.2.6 Volume Ballooning

In some cases, we may want to change the parameters of the energy function to obtain a large enough tumor segmentation, but changing the relative weight between the data term and the smoothness term may be insufficient to obtain larger segmentation results. This motivates us to incorporate volume ballooning into our approach to get larger tumor segmentation.

We adopt uniform ballooning [9] in our method. It is implemented by adding a bonus value in the data term for each pixel in the image if it is labeled as the foreground. For each pixel p, we add a term $B_p(0) = \beta$ as an additional cost for being labeled as the background. Then, the energy function, based on the approach adopted in [9], with volume ballooning becomes:

$$E(\boldsymbol{f}) = \sum_{p \in \Omega} D_p(f_p) + \lambda \sum_{pq \in \mathcal{N}} V_{pq}(f_p, f_q) + \sum_{p \in \Omega} B_p(f_p) \tag{2.28}$$

Because B_p is the volumetric bias and an unary (linear) term, it is convenient to be included and optimized by graph cuts. The larger the parameter β is, the larger the ballooning force is, where β is determined by validation.

Chapter 3

Automatic Brain Tumor Segmentation

Brain tumor segmentation is important in diagnosis and treatment of brain tumors. We investigate automatic brain tumor segmentation by combining deep convolutional neural networks with regularization by a graph cut.

In this chapter, we present the details of our approach. First, we discuss how we apply several deep convolutional networks for brain tumor segmentation. Then, we describe how we apply graph cuts with different data terms to regularize the segmentation results obtained by neural networks.

3.1 Overview of our approach

First, we evaluate several different deep convolutional network architectures for the task of tumor segmentation. We train the networks on overlapping brain crops. We first tested on whole brain slices using the network trained on whole brain slices but it did not work well. Since the tumor pixels account for a very small portion in the whole brain slice, segmenting tumors from background is a highly imbalanced dense prediction task. We use the loss function that takes the imbalance of the training data into consideration by putting a higher weight for the loss incurred by the pixels whose true label is tumor. The next step is to combine the results on overlapping crops to obtain segmentation on a whole brain slice. We do this by a simple union operation.

Second, we apply graph cut regularization using the segmentation probability map obtained by deep convolutional neural networks to further improve the segmentation results obtained by deep neural networks. We explore and evaluate several ways in which the tumor probabilities learned by the neural networks can be converted to the data terms required by regularization a graph cut. We optimize all the parameters of the graph cuts segmentation by using grid search on the validation data.

3.2 Deep CNNs for tumor segmentation

Since our labeled data is limited and a brain tumor is imbalanced in the whole slice, we crop the whole slices into overlapping small crops, as shown in Figure 3.1.

We experimented with different crop sizes and found through the experiments that training on crops of size 128 × 128 is faster than larger crops and works comparably to training on larger crops. The crop size should consist of the factor that is the power of 2 because each pooling layer will decrease the resolution by half. The image crop is overlapped by half in one dimension with previous one. The experiment chapter shows how the crop size affects the accuracy of the segmentation. We make the crops overlap since any particular crop can contain only a part of a tumor, resulting in poor tumor detection. With overlap, a neighboring crop will have a larger or almost whole tumor, likely to give a better tumor detection. Then, we divide the whole dataset into training, validation and test sets. We choose and experiment with the deep convolutional neural network architectures that were previously successful for semantic segmentation and medical image segmentation, such as fully convolutional neural networks [36], deep dilated residual networks [58] and U-Net [46] on training data and validate its generalization performance on validation data. When making segmentation predictions on test data of whole brain slices, we still first predict the segmentation on overlapping crops that are exactly the same size and overlap as those used during training, i.e., by running forward propagation and getting the prediction score or probability map, then taking its maximum element index for each pixel as the segmentation prediction label. Then, we combine the segmentation results on the overlapping crops by a simple union operation.

3.2.1 Several Deep CNN Structures

In this section, we describe several convolutional neural network structures that we adopted in this thesis, including fully convolutional neural networks (FCNs) [36], dilated residual networks (DRNs) [58] and U-Net [46]. Current state-of-the-art approaches for semantic image segmentation all employ some form of fully convolutional network, which is the first work to train FCNs end-to-end for pixelwise prediction as stated in [36] and can be seen as the baseline network for the tumor segmentation task. U-Net is widely used and achieved excellent performance for medical image segmentation. Dilated residual networks are the dilated version of the residual networks which are the state-of-the-art performance network architectures on ImageNet. These networks are all adaptable for brain tumor segmentation.

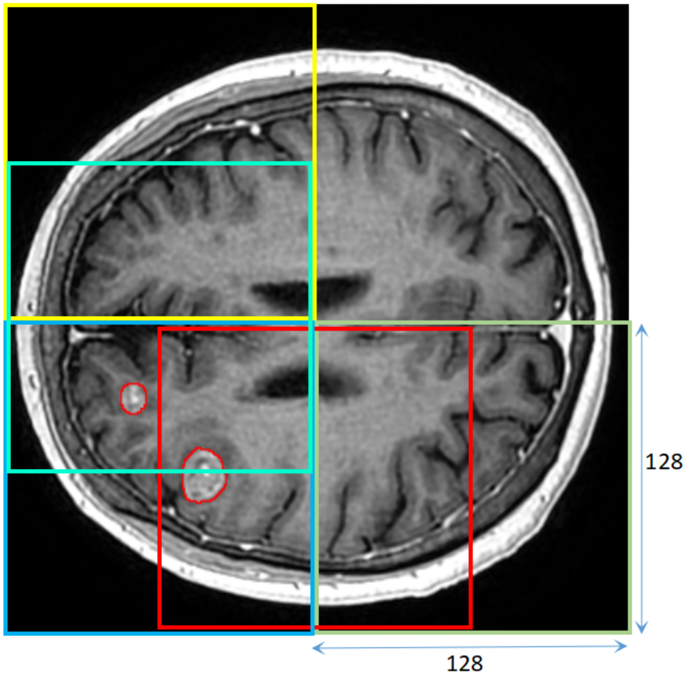


Figure 3.1: Examples of overlapping crops. The boxes with different colors are the corresponding crops on the original image.

Fully Convolutional Network

A fully convolutional network (FCN) [39] is the network for which the learnable layers are only convolutional layers, and does not have any fully connected layers. There are several advantages of FCN over convolutional networks with several fully connected layers. First, we can use various image sizes as input to the FCN, while convolutional networks with fully connected layers can only accept input images with a fixed size. Second, fully connected layers need lots of memory storage and numerical computation because lots of parameters need to be learned, while convolutional networks can learn good features and require much less parameters. Fully connected layers can be converted into convolutional layers by replacing fully connected layers with convolutional layers. Then, the whole network becomes fully convolutional.

While fully convolutional networks can be modified to perform semantic segmentation, their segmentation results are coarse. This issue is addressed by adding skips [6] that combine the final prediction layer with lower layers. This turns a line structure of convolutional neural network into a directed acyclic graph with edges that skip from lower layers to higher ones. Combining fine layers with coarse layers allows the model to fuse different levels of information, as suggested in [36].

We adopt the VGG 19-layer net [49], but modify it to perform tumor segmentation. We remove three convolutional layers with 512 filters to improve efficiency and reduce overfitting. The size of the input image to the FCN is 128 × 128 × 1. We discard the final classifier layer, and convert all fully connected layers to convolutional layers. We append a 1 × 1 convolution with channel dimension 2 to predict the classification scores for the tumor and background classes at each of the coarse output location. Then, the network is followed by a deconvolution layer to upsample the coarse outputs to pixel dense outputs and with skip layers to fuse the information with earlier layers as described in [36]. We reduce the two layers with 4096 filters to the two layers with 2048 filters, which can improve efficiency and reduce overfitting. We also adopt dropout [50], a regularization technique when training deep neural networks, to avoid overfitting as in [36]. The final output volume size is 128 × 128 × 2, where it contains the prediction score for tumor and background at each pixel. The modified fully convolutional network architecture is shown in Figure 3.2.

U-Net

U-Net [46] is a popular architecture that is widely used in the biomedical community [23]. Its main idea is to supplement a usual convolutional network by successive layers which gradually upsample the previous layers. In order to fuse different levels of information, high resolution features representation from the contracting path are combined with the upsampled feature maps. These combined layers can then learn context information. The upsampling part allows the network to propagate context information to higher resolution layers. As a consequence, the expansive path and contracting path yields a u-shaped architecture.

We modify original U-Net for tumor segmentation. The original U-Net only uses the valid part of each convolutional layer, i.e., the segmentation map only contains the pixels whose full context is available in the input image. Thus, the final output segmentation map resolution is smaller than the input image. To predict the labels of all the pixels in the input image, original U-Net has to use an overlap-tile strategy. To predict the labels of the pixels in the border region of the image, the missing context is extrapolated by mirroring the input image. But it is not convenient and the overlap-tile strategy is not suitable especially when the tumors are around the boundary, so we use zero padding around the input boundary before applying the convolutional operation so that the output size is the same as the input size. Thus, the final output resolution is the same as the size of the original input image. The input image size is 128 × 128 × 1 and the output segmentation map size is 128 × 128 × 2 by adding zero padding. The overlap-tile strategy becomes unnecessary and we can directly predict all the pixel labels by once forward propagation without mirroring the input image. In addition, we adopt dropout [50], a regularization technique when training deep neural networks, to avoid overfitting as in [46]. Also, we remove seven convolutional layers to improve efficiency and add seven batch normalization layers to make it easier to train the U-Net. The modified U-Net architecture is

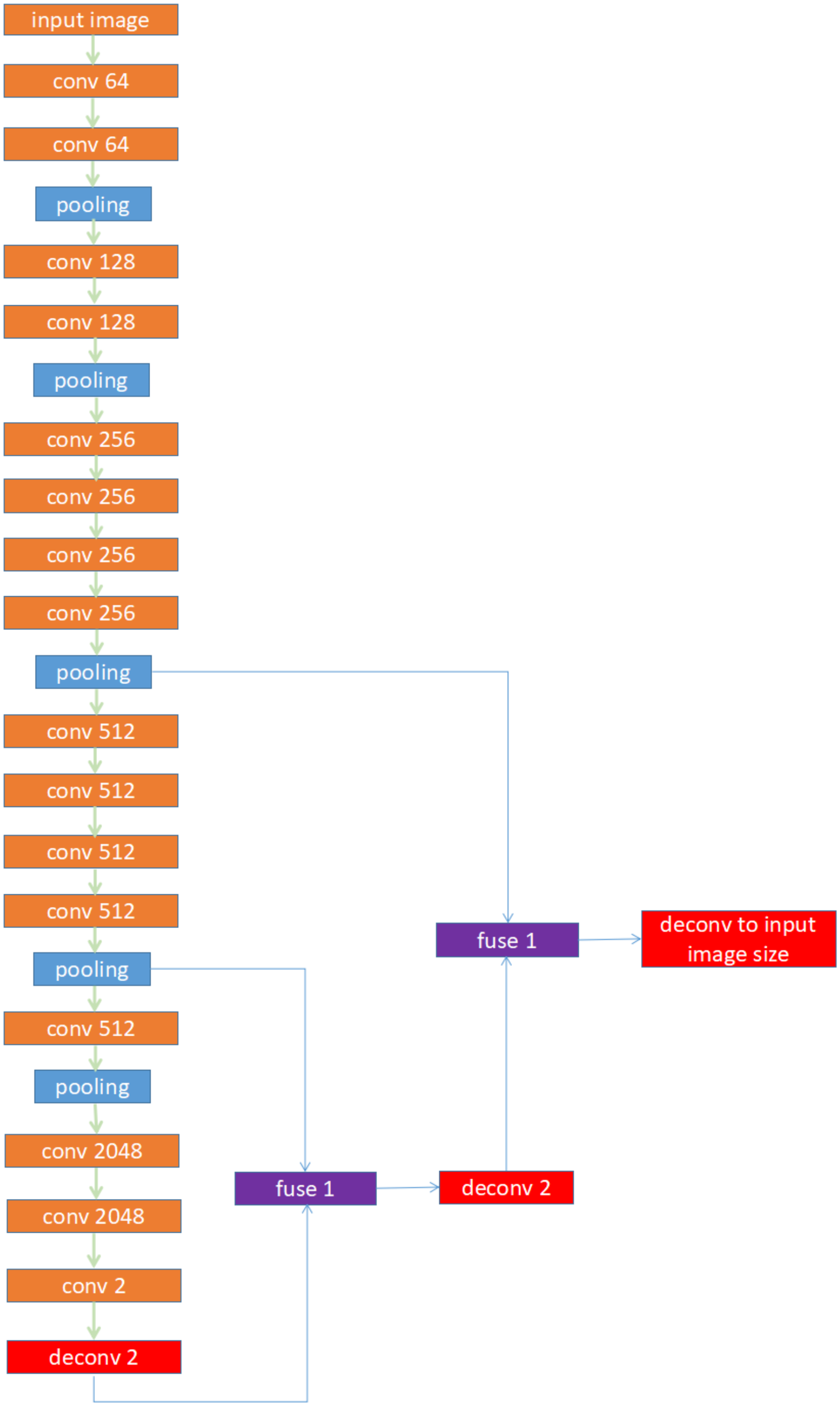


Figure 3.2: Modified fully convolutional neural network architecture

shown in Figure 3.3.

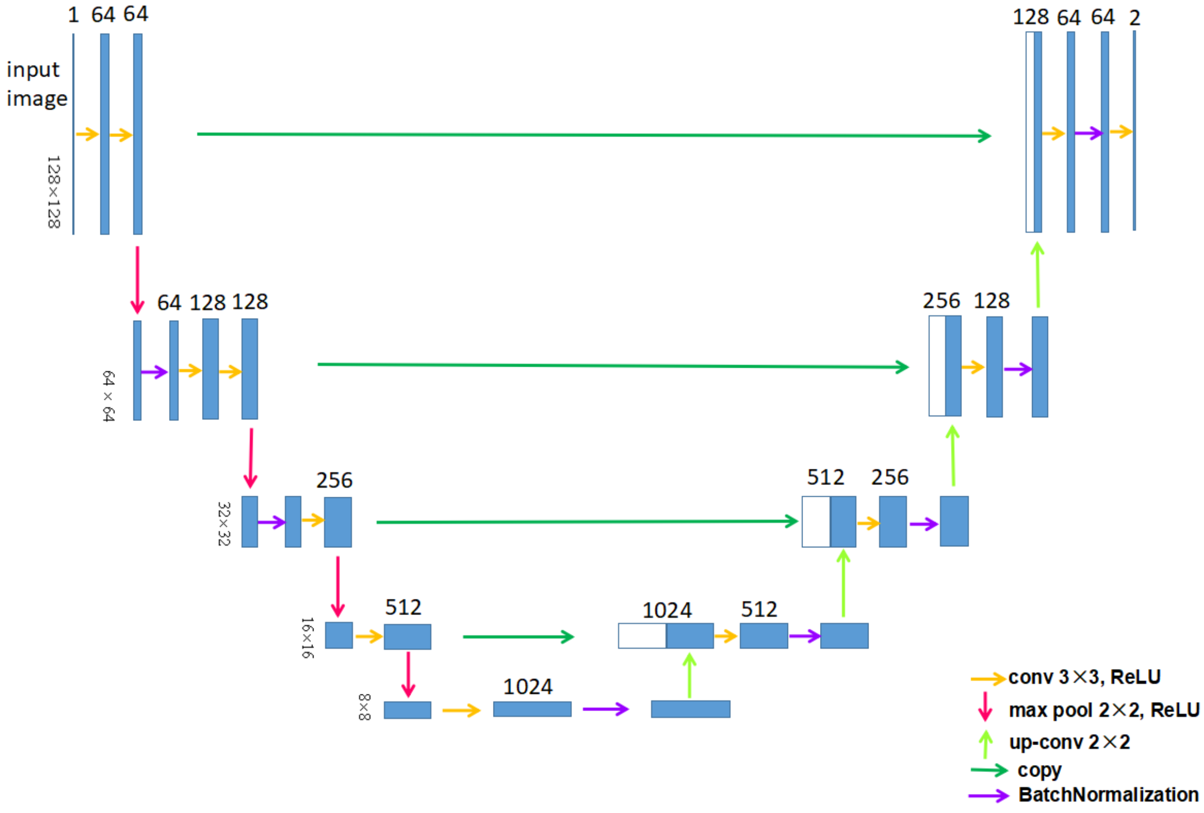


Figure 3.3: Modified U-Net architecture

Dilated Residual Networks

The residual networks are the state-of-the-art network architecture for image classification [18]. Residual network building block is shown in Figure 3.4. The operation $\mathcal{F} + x$ is performed by a shortcut connection and element-wise addition. Batch normalization, described in Section 2.1.4, is adopted right after each convolution operation and before the Rectified Linear Unit (ReLU) activation. Dropout is not used in the network.

Residual networks can be converted into dilated residual networks (DRNs) by dilated convolution [57], which is described in Section 2.1.2. DRNs architecture is shown in Figure 3.5. Dilated residual networks [58] yield improved image classification performance and can be transformed to perform tumor segmentation. Due to the downsampling of original DRNs, various upsampling techniques, e.g., bilinear upsampling and deconvolution [44, 10], are necessary in the following layers to get full resolution output for tumor segmentation. The predictions synthesized by the output layer are upsampled to full resolution using bilinear interpolation to perform tumor segmentation by dense prediction. The input image is sequentially processed by each residual network building block with dilated convolution. The size of the input image crops is 128 × 128 × 1 and the size of the output segmentation map is 128 × 128 × 2.

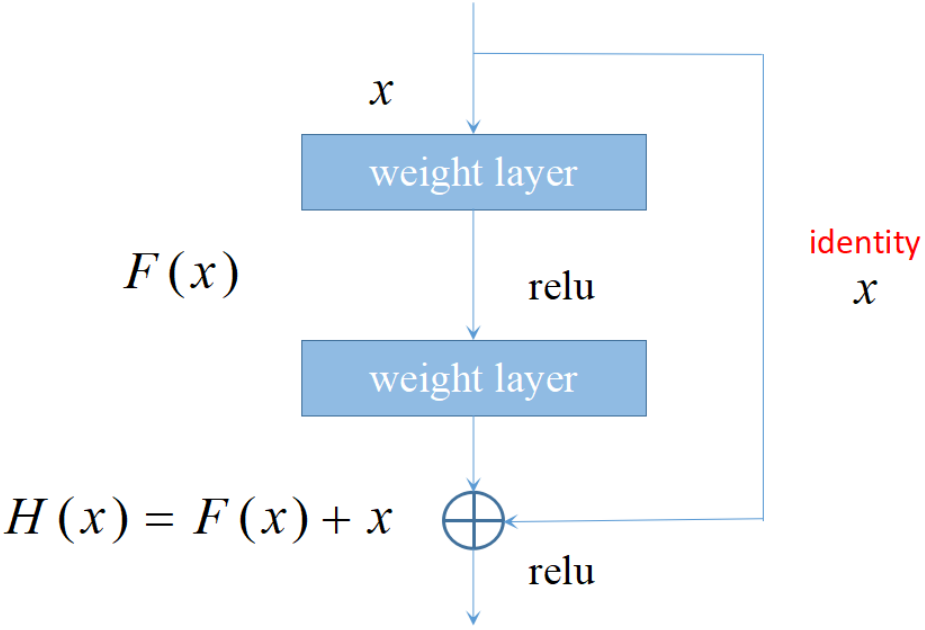


Figure 3.4: Residual learning: a building block.

3.2.2 Data Augmentation

One way to make the deep neural network generalize better is to train it on more data due to large number of parameters. Since our labeled training data of brain volumes is limited and training the whole network needs lots of labeled data, we adopt simple data augmentation, e.g., horizontal and vertical flipping of original images. We add these augmented data into the training and validation dataset, tripling the number of training and validation images. No other data augmentation is used. Other data augmentation is also possible, e.g., subcrops of the original images, but the performance improvement is not obvious and training time is much longer.

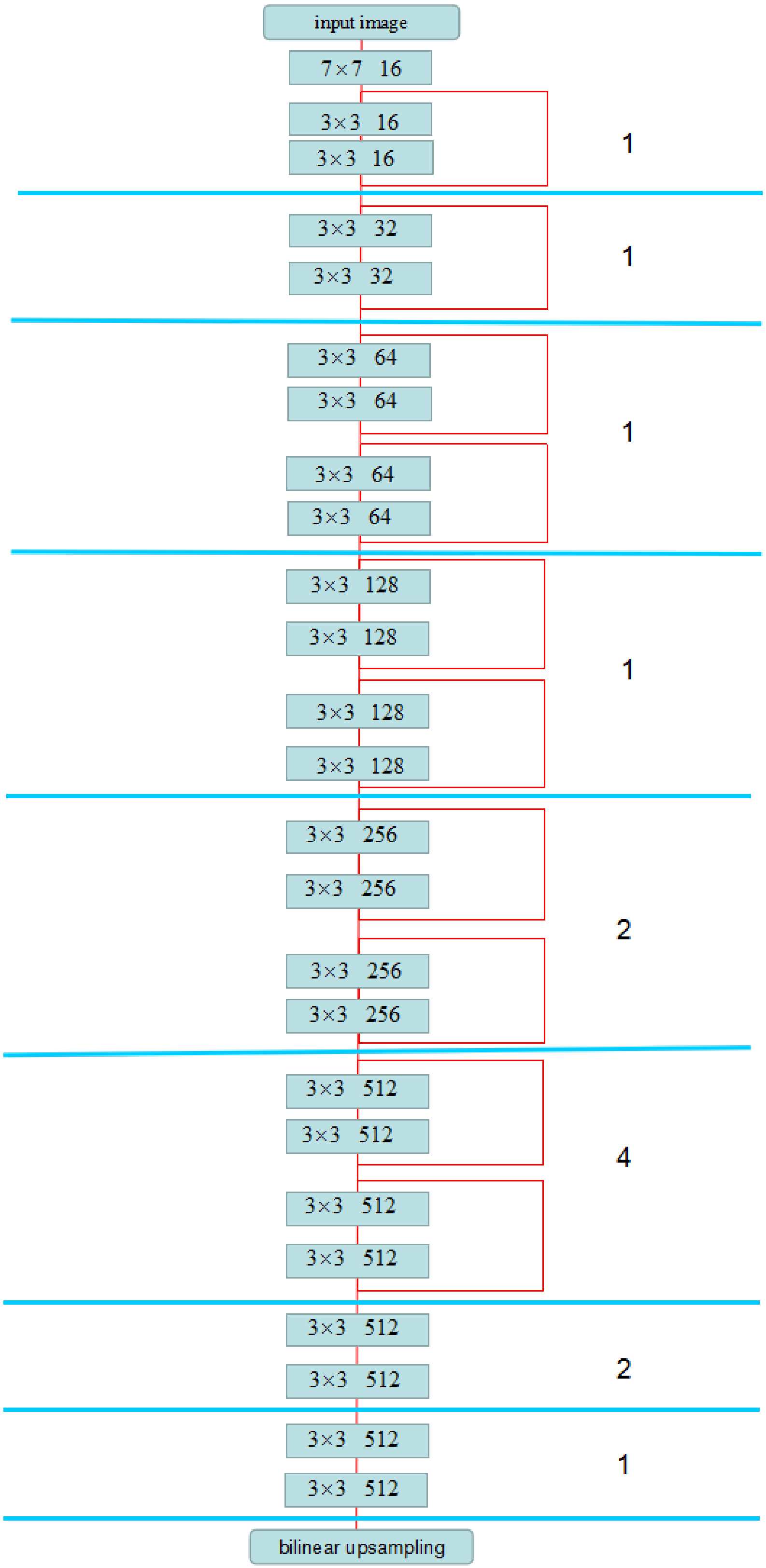


Figure 3.5: Dilated residual networks architecture. Each rectangle is a Convolution-Batch normalization-Relu activation group and the numbers mean the number of filters in that layer. The red connection besides the rectangle is the skip connection. And the number between two blue lines is the dilation rate for the corresponding convolutional layers.

3.2.3 Training from scratch vs. utilizing pretrained model

In real applications, very few people will train the whole neural network from scratch with random parameter initialization since it is not easy to obtain sufficient size of labeled data to train the whole network and needs very long training time by training the network on a large dataset from scratch. On the other hand, it is common to adapt various pretrained deep CNNs, e.g., VGG [49] and residual networks [18], on a very large dataset, e.g., ImageNet which contains 1.2 million images with 1000 classes, for some specialized computer vision tasks [36, 13, 45].

There are two major ways using these pretrained models [28]:

(1) **Convolutional network as a fixed feature extractor:**

Take a deep CNN pretrained on ImageNet, then remove the last fully-connected layer with 1000 units (this layer predicts scores for the 1000 classes). Next, use the rest of the convolutional network as a fixed feature extractor for the new dataset.

(2) **Fine-tuning the convolutional network:**

Fine-tuning the neural network means that the weights of the pretrained network are also updated on the new dataset. We can either fine-tune the weights of all the layers of the pretrained deep CNNs, or keep the parameters of some earlier layers fixed and only fine-tune some higher-level layers of the pretrained network. The motivation is that the features extracted by the earlier layers of deep CNNs contain more generic features, e.g., edge or color blobs, that should be useful for more general tasks, but later layers of deep CNNs become more specific to the specialized tasks. For example, it is only necessary to update the layers from conv3_1, which is the first convolutional layer in the third convolution block of VGG16 [49], and up layers of VGG16 trained on ImageNet, for object detection using fast R-CNN [13] since the features extracted by the earlier layers of VGG16 is generic and task independent.

In this thesis, we do not adopt any pretrained model for several reasons. First, we only need to predict for each pixel to be one of the two classes, i.e., tumor or background, instead of many classes, so our task does not need the amount of training data as much as ImageNet. Second, the voxels in our image data are represented by 16-bit integers, while most pretrained models are trained on 8-bit images. The pixel intensity range of the input images for the pretrained models is different from our brain image data, so the pretrained models may not be applicable to our data. Third, our medical image data is drastically different from the real world images from ImageNet, so the learned features on ImageNet are not useful for our task. Therefore, we train all deep neural network models from scratch and parameter initialization strategies are described in Section 3.2.4.

3.2.4 Parameter initialization

Training algorithms for deep learning models are usually iterative. These algorithms include SGD, NAG, Adam, etc, as described in Section 2.1.5, and require some initialization strategies to specify initial point for the optimization algorithms to begin the iterations. Moreover, training deep models is a difficult task. The initialization point strongly affects the performance of most algorithms and can even determine whether the training algorithm converges at all. When the training algorithm does converge, the initial point can determine how fast the learning algorithm converges and whether it converges to a point with a high or low loss function value.

Initialization strategies for training deep neural networks are simple and heuristic. Designing improved initialization strategies is a difficult task because optimizing the deep neural network is not well understood. Another difficulty is that from the viewpoint of optimization, some initial points may be better but from the viewpoint of generalization performance, these initial points may not be good choices. How the initial point affects generalization is especially unclear. Almost no useful guidance is available for how to choose the initial point. Even though we do not have useful guidance, it is common to initialize all the weights of the deep neural network by randomly drawing from a Gaussian or uniform distribution or their variants motivated by enabling each unit to compute a different function.

In this thesis, we adopt initializations that draw samples from a truncated normal distribution centered on 0 with standard deviation $\sigma = \sqrt{\frac{2}{num_u}}$ where num_u is the number of input units in the weight tensor.

3.2.5 1 slice vs 3 slices

Another variation of CNNs that we have tried is to train the network on three slices at a time, rather than one slice. The intuition behind training the neural networks on three slices instead of one slice is that the crops with three slices contain more tumor spatial and context information, and may be utilized for better differentiating the tumor pixels from the background pixels.

For training and validating on three slices, we take the previous, the current, and the next slice and concatenate them together as images with three channels. We use the ground truth of the middle slice as the ground truth. In other words, the previous and next slice are used only for additional context to classify pixels in the current slice. Other training process and experimental settings are the same.

3.2.6 Balanced Loss Function

Since tumor pixels account for a very small portion in the whole slice image, segmenting a tumor from background is a highly imbalanced dense prediction task. To further deal with the imbalanced class problem, we use a class-balanced cross-entropy loss function for training deep neural networks. We denote the parameters of all the network layers as **W**, and consider the objective function that is used in [56]:

$$\mathcal{L}(\mathbf{W}) = -\beta \sum_{j \in Y_+} logPr(y_j = 1|X; \mathbf{W}) - (1-\beta) \sum_{j \in Y_-} logPr(y_j = 0|X; \mathbf{W}) \tag{3.1}$$

The loss function is computed over all pixels in a training image $X = (x_j, j = 1, ..., |X|)$, Y_- and Y_+ denote the background pixels and tumor pixels respectively. Where β controls the relative importance between the background pixels and tumor pixels. $Pr(y_j = 1|X; \mathbf{W}) = \sigma(a_j) \in [0, 1]$, is computed by sigmoid function $\sigma(\cdot)$ on the activation value at pixel j. We also investigate the effects of β on the segmentation performance.

3.3 Graph cut regularization

After obtaining the segmentation results by deep CNNs, we also get the segmentation map with probability of being tumor for each pixel. Then, we apply graph cut regularization with different data terms to improve the deep neural network segmentation results. We design and evaluate several data terms that aim to alleviate the segmentation artifacts by CNNs. For example, the precision of the segmentation boundary by CNNs is often lacking due to the max pooling layers that reduce resolution, resulting in imprecise boundary. Graph cut regularization can help to align tumor boundary to intensity edges in the image that can improve results since tumor edges often coincide with intensity edges.

3.3.1 Data Term

The data term measures how well the label assigned to a pixel fits into the image data. For binary segmentation, it indicates the cost of assigning pixel p to the tumor and background labels. We design and evaluate various data terms. In the rest of this section, we describe these data terms. Let p be an image pixel and let $f(p)$ be the output of the CNNs. During training, $f(p)$ is interpreted as the probability of being a tumor for pixel p and therefore $0 \leq f(p) \leq 1$ for any pixel p of the image.

(a) First, we design perhaps the simplest possible data term that only considers pixels for which CNNs predict to be tumor (or background) with high probability. The intuition behind this data term is that pixels with high probability of being tumor (or background) should be assigned to tumor (or background) for sure during graph cut optimization. The rest of the pixels should ignore the probability learned by CNNs since this probability, being far from confident, may be unreliable. Thus any pixel whose probability of being tumor (or background) is not high enough gets equal score for being either tumor or background, then the learned probabilities are ignored. We use two different parameters, for the tumor and the background to determine if the probability is high enough. If for pixel p, $f(p) \geq \delta_t$, then pixel p is hard constrained to be labeled as tumor by assigning a prohibitively large data cost K to pixel p taking label 0. Similarly, if for pixel p, $1 - f(p) \geq \delta_b$, then pixel p is hard constrained to be labeled as background by assigning a prohibitively large data cost K to pixel p taking label 1.

More specifically, if $f(p) > \delta_t$, then

$$D_p(1) = 0 \quad \textit{and} \quad D_p(0) = K \tag{3.2}$$

If $1 - f(p) > \delta_b$, then

$$D_p(1) = K \quad \textit{and} \quad D_p(0) = 0 \tag{3.3}$$

If neither $f(p) > \delta_t$ nor $1 - f(p) > \delta_b$, then

$$D_p(0) = D_p(1) = 0 \tag{3.4}$$

We search for δ_t and δ_b on the validation data and also ensure $min\{\delta_t, \delta_b\} > 0.5$ since the idea is to select pixels that are confident in their labels from CNNs for hard constraining.

(b) We empirically observe that the pure probabilities of being tumor learned by CNNs do not spread out uniformly between 0 and 1. Most values are concentrated around 0 and 1, and the rest are very sparse. Therefore, we design the data term which is the same as (a) but based on log probability, rather than probability itself. The log function spreads out values more uniformly, and, therefore, it is easier to find the threshold and searching the parameters does not have to be finely tuned. For pixels that have high log likelihood as being a tumor (or background) should be assigned to tumor (or background) for sure during graph cut optimization. The rest of the pixels should ignore the log likelihood learned by CNNs since this log likelihood, being far from confident, may be unreliable. Thus any pixel whose log likelihood of being tumor (or background) is not high enough gets equal score for being either tumor or background, then the learned log likelihood is ignored. We use two different parameters, for the tumor and the background to determine if the log likelihood is high enough. If for pixel p, $log(f(p)) \geq \delta_{logt}$, then pixel p is hard constrained to be labeled as tumor by assigning a prohibitively large data cost K to pixel p taking label 0. Similarly, if for pixel p, $log(1 - f(p)) \geq \delta_{logb}$, then pixel p is hard constrained to the background by assigning a prohibitively large data cost K to pixel p taking label 1.

More specifically, if $log(f(p)) > \delta_{logt}$, then

$$D_p(1) = 0 \quad and \quad D_p(0) = K \tag{3.5}$$

If $log(1 - f(p)) > \delta_{logb}$, then

$$D_p(1) = K \quad and \quad D_p(0) = 0 \tag{3.6}$$

If neither $log(f(p)) > \delta_{logt}$ nor $log(1 - f(p)) > \delta_{logb}$, then

$$D_p(0) = D_p(1) = 0 \tag{3.7}$$

We search for δ_{logt} and δ_{logb} on the validation data and also ensure $min\{\delta_{logt}, \delta_{logb}\} > log0.5$ since the idea is to select pixels that are confident in their labels from CNNs for hard constraining.

(c) When we obtain the label for each pixel, we treat all background pixels predicted by CNNs in the same way and all tumor pixels in the same way. If they are classified as background by CNNs, they will have constant cost for being tumor and background for graph cut optimization. If they are classified as tumor, they will have constant cost for being tumor and background. If any pixel that is classified as tumor by deep CNNs has data cost of 0 to be assigned to tumor and data cost of 1 to be background; if any pixel that is classified as background has data cost of 0 to be assigned to background and data cost of 1 to be tumor. The intuition behind this data term is that we do not treat probabilities estimated by CNNs as exact, but rather treat them as classification labels into tumor or background. Unlike (a) and (b), here we do not use hard-constraints and do not ignore pixels with low confidence in their class labels.

More specifically, for any pixel that is classified as tumor, then

$$D_p(1) = 0 \quad and \quad D_p(0) = 1 \tag{3.8}$$

For any pixel that is classified as background, then

$$D_p(1) = 1 \quad and \quad D_p(0) = 0 \tag{3.9}$$

(d) If any pixel is predicted as tumor with high probability, then the cost for being tumor will be low and the cost for being background will be high; if any pixel is predicted as tumor with low probability, then the cost for being tumor will be high and the cost for being background will be low.

More specifically, we use the negative log likelihood of being tumor and background as the data term to make the probability value spread out, as shown in the following:

$$D_p(1) = -log(f(p)) \quad and \quad D_p(0) = -log(1 - f(p)) \tag{3.10}$$

(e) Intuitively, the cost for being tumor and background for each pixel not only relates to its probability predicted by CNNs but also relates to the distance between one specific pixel and segmentation boundary by CNNs. This is because the segmentation boundary produced by CNNs is not reliable. Due to the max pooling layers, resolution of the original image is reduced, resulting in less precise boundary localization. We use the negative log likelihood of being tumor and weighted distance transform to construct the data term.

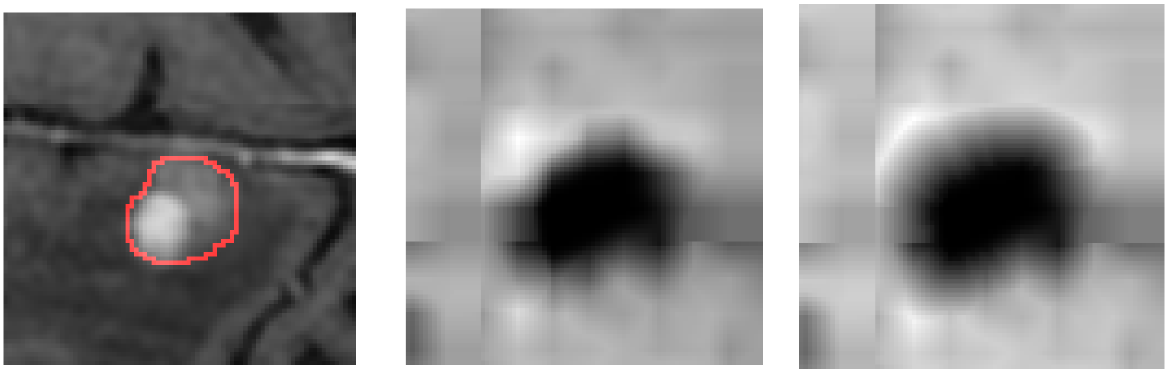
More specifically, for any pixel, the cost for being tumor will be the smaller value between the negative log likelihood of being tumor and weighted distance between that specific pixel and the segmentation boundary by CNNs, as shown in Equation 3.11; the cost for being background will be the smaller value between the negative log likelihood of being background and weighted distance between that specific pixel and the segmentation boundary by CNNs, as shown in the following:

$$D_p(1) = \min(-log(f(p)), d * w_1) \tag{3.11}$$

and

$$D_p(0) = \min(-log(1 - f(p)), d * w_2), \tag{3.12}$$

where w_1 and w_2 are determined by searching on the validation data. And d is the distance to the segmentation boundary and is calculated by distance transform, which is calculated as the distance between the nonzero pixels and the boundary between nonzero pixels and zero pixels. The visualization of an example of data term (e) is shown in Figure 3.6.



(a) brain image crop, the red contour is the ground truth boundary.

(b) the foreground cost by negative log likelihood.

(c) the foreground cost by negative log likelihood and weighted distance transform of data term (e) when $w_1 = 1$, as in Equation (3.11).

Figure 3.6: Visualization of data term (e). The vertical and horizontal lines in (b) and (c) are the crop boundary.

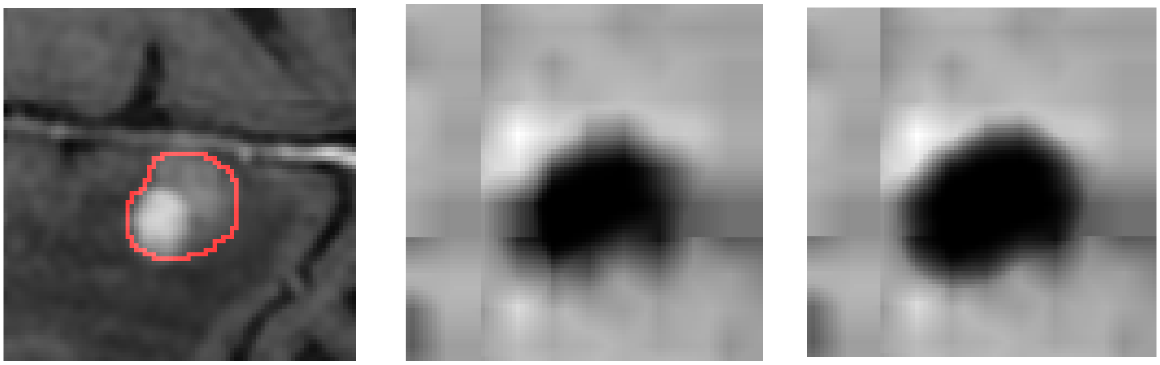
(f) Here is another variant of the idea in (e). We want the data terms for pixels not far from the segmentation boundary to have lower weight comparatively to data terms for pixels farther away from the segmentation boundary. We use distance transform to calculate the distance to the segmentation boundary by deep CNNs, denoted by d. Then we set a threshold denoted by r and use the weight denoted by w. The pixels that are near the segmentation boundary have smaller weight and the pixels that are far away from the segmentation boundary have larger weight; then the pixels around the segmentation boundary by CNNs have lower corresponding data cost. The weight w and the weighted negative log likelihood data term are shown in the following:

$$w = \begin{cases} \frac{d}{r} & d < r \\ 1 & \text{otherwise} \end{cases} \tag{3.13}$$

and

$$D_p(1) = -w \cdot log(f(p)) \quad and \quad D_p(0) = -w \cdot log(1 - f(p)) \tag{3.14}$$

The visualization of an example of data term (f) is shown in Figure 3.7.



(a) brain image crop, the red contour is the ground truth boundary.

(b) the foreground cost by negative log likelihood.

(c) the foreground cost by weighted negative log likelihood of data term (f) when $r = 10$, as in Equation (3.14).

Figure 3.7: Visualization of data term (f).

3.3.2 Smoothness Term

Smoothness term in the energy function is a piarwise term encouraging the coherence between neighboring pixels and can be thought of as cost for the discontinuity, i.e., different labels between neighbouring pixels. Hence, the penalties are all on the boundary of the segmentation. Since our goal is to minimize the energy function, this smoothness term encourages the segmentation to have a shorter boundary. The smoothness term we use is as the following:

$$V_{pq}(f_p, f_q) = w_{pq} \cdot \delta(f_p, f_q), \tag{3.15}$$

where

$$\delta(f_p, f_q) = \begin{cases} 1 & f_p \neq f_q, \\ 0 & otherwise. \end{cases}$$

If neighboring pixels are assigned the same label, there is no penalty. When $w_{pq} = constant$, the cost for any pair of pixels that are assigned to different labels is the same and independent of how strong the intensity contrast between these pixels. To indicate the difference between the different neighbouring pixels, w_{pq} is usually set as a non-increasing function of $|I_p - I_q|$. The cost w_{pq} is often based on local intensity gradient, the Laplacian zerocrossing, the gradient direction and some other criteria. In this thesis, we use the weight determined as:

$$w_{pq} = exp(-\frac{(x_p - x_q)^2}{2\sigma^2})\frac{1}{dist(p, q)}, \tag{3.16}$$

where $dist(p, q)$ is the distance between the pixel p and pixel q and σ is set to be the average absolute intensity difference between neighboring pixels in the image as in [35, 9].

We have different sampling rates along dimensions x, y and dimension z. $dist(p, q)$ for two pixels on the same slice is different from those on different slices. We adopt 6-neighborhood system in this thesis, as indicated in Figure 2.9.

Chapter 4

Experimental Results

In this chapter, first we discuss the details of our experimental image data. Secondly, we describe the evaluation metrics that are used for evaluating the segmentation results. Thirdly, we present the parameter setting when training the deep CNNs. Next, we present the segmentation results by different deep convolutional networks. Then, we investigate the effects of different weight of the balanced loss function for training deep CNNs. After that, we compare the segmentation results by only using deep CNNs and combining deep CNNs with graph cut regularization. Then, we compare the segmentation results by only using single slice and three slices. Finally, we show the effects of the crop size on the segmentation results.

4.1 Image Data

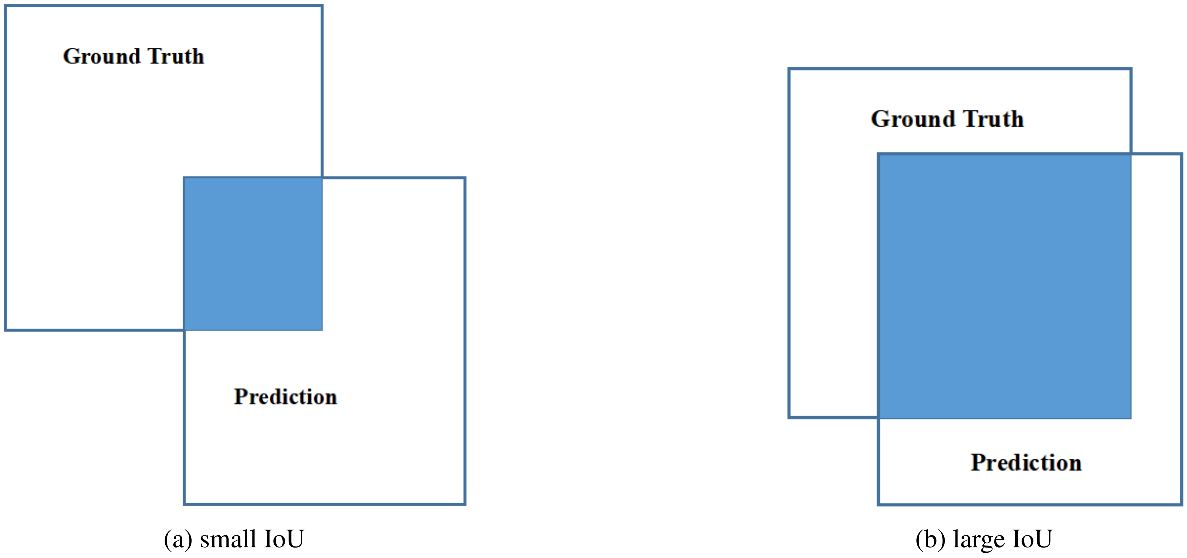
Our data is provided by Aaron Ward and Glenn Bauman from Lawson Health Research Institute. There are 64 MRI brain scans from 27 patients. Some scans are from the same patient in different stages. The training set contains 30 volumes with 9098 crops. The validation set contains 8 volumes with 2350 crops. The test set contains 7 volumes with 148 slices. We adopt simple data augmentation only on training and validation data, i.e., horizontal and vertical flipping of the image crops. We add these augmented data into the training and validation data, tripling the number of training and validation images. No other data augmentation is used. Thus, by data augmentation, the training set contains 27294 image crops and the validation set contains 7050 image crops. The crop size is 128×128 pixels. We also investigate the effects of different crop size on the segmentation performance in Section 4.8. The voxels are represented by 16-bit integers.

4.2 Evaluation Metric

The commonly used evaluation metrics in semantic image segmentation are pixel accuracy and intersection over union (IoU). Let n_{ij} denote the number of pixels of class i predicted to be class j. Assume there are N_c different classes, and let $t_i = \sum_j n_{ij}$. IoU is the intersection of pixels predicted to be certain class and the ground truth pixels divided by the union of the pixels predicted to be certain class and ground truth pixels. Mean IoU is the average of IoU over all classes:

$$M_{iou} = (1/N_c) \sum_i n_{ii}/(t_i + \sum_j n_{ji} - n_{ii}) \tag{4.1}$$

Figure 4.1 shows two examples of intersection over union and indicates that the larger the IoU is, the better the segmentation results are.



(a) small IoU

(b) large IoU

Figure 4.1: Examples of intersection over union, which is calculated by the intersection, indicated by the blue area, divided by the union of the two rectangle.

Pixel accuracy means the total number of correctly predicted pixels divided by the total number of pixels:

$$Pixel_{accuracy} = \sum_i n_{ii}/ \sum_i t_i \tag{4.2}$$

4.3 Experiment setting

The parameter initialization strategies adopted are presented in Section 3.2.4. We train all the deep neural networks by Adam [30], as described in Section 2.1.5, an adaptive first-order gradient-based optimization algorithm. The networks are trained in 40 epochs with initial learning rate 0.0001, $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The minibatch size is 2 image crops. We also adopt early stopping, i.e., if there is no improvement after 10 epochs on the validation data, the training process will be stopped. We decrease the learning rate by multiplying a factor of 0.3 when the validation loss has no improvement for 5 epochs. Unless otherwise specified, we use cross entropy as default loss function and use Tensorflow and Keras, which are open-source machine learning library, to implement the deep CNNs in Python. Our experiments are done in computers with 4GB GPU memory and 16 GB RAM (main memory).

4.4 Visualization of segmentation results with different deep Convolutional Neural Networks

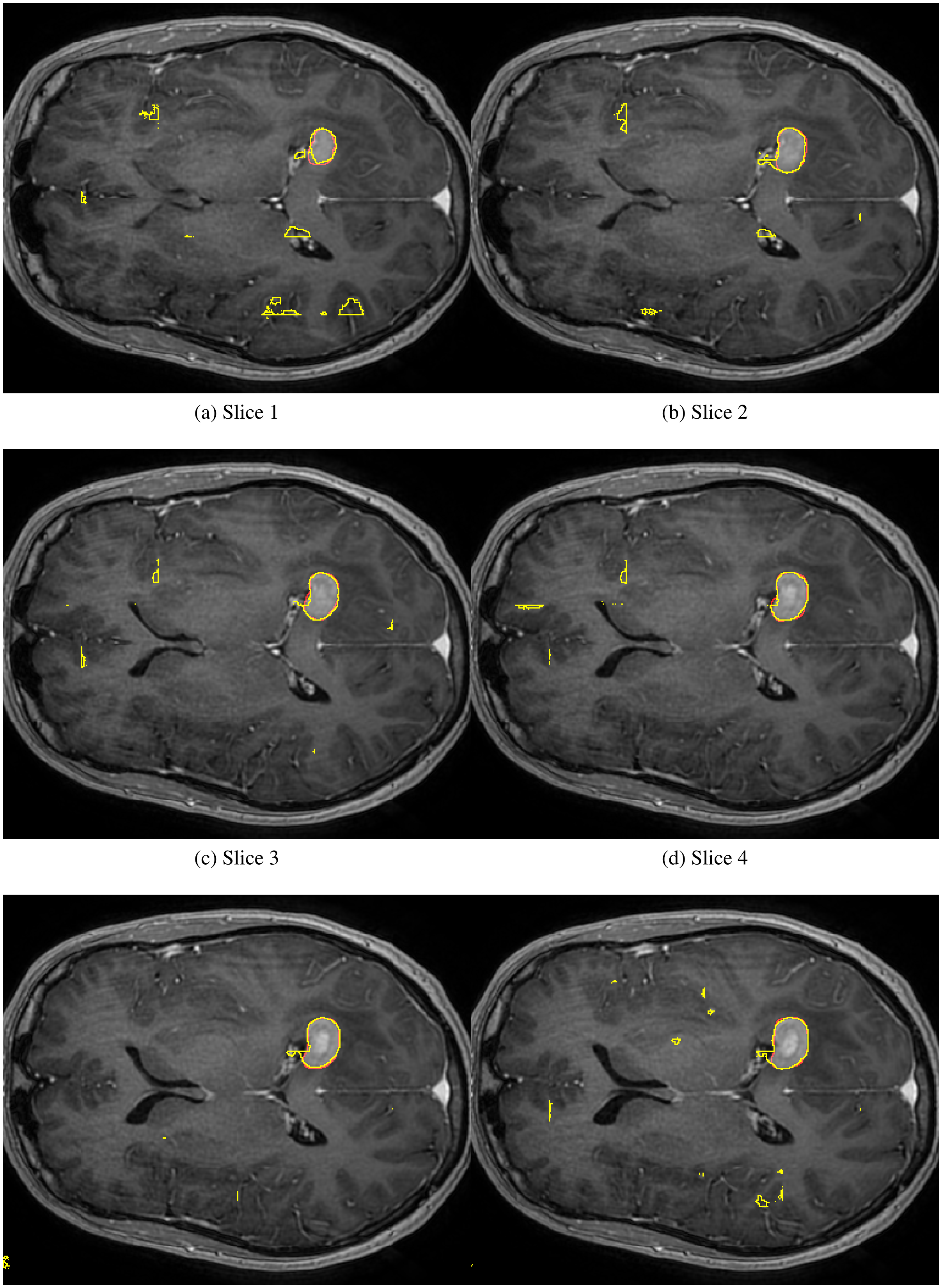
In this section, we present some segmentation results by different deep CNNs. The evaluation metrics for evaluating the performance of different deep neural networks which are described in Section 3.2.1, including Fully Convolutional Networks (FCNs) [36], U-Net [46] and Dilated Residual Networks (DRNs) [58], are shown in Table 4.1, where intersection over union (IoU) is calculated for tumor class. From this table, we can see that U-Net performs slightly better than DRN. Pixel accuracy measure is almost as accurate for all networks, but it hides the fact that a lot of pixels classified as tumor are actually mistakes in the case of FCN. This is because the pixel accuracy measure is dominated by the background pixels that are in the overwhelming majority.

From the visualization of the segmentation results by different networks in the rest parts of this section, we can see that the segmentation results by FCN are quite noisy. FCN often predicts incorrectly for tumor and background pixels and the segmentation boundary by FCN is not as smooth as U-Net and DRN. The segmentation results by U-Net and DRN are similar and comparable, but the segmentation boundary by U-Net is slightly smoother and U-Net fits the boundary of tumor slightly better than DRN.

deep CNNs	Pixel accuracy	IoU
FCN	0.9952	0.6477
U-Net	0.9973	0.7286
DRN	0.9972	0.7224

Table 4.1: Evaluation metrics for the segmentation performance of different deep convolutional neural networks.

Fully Convolutional Neural Network The segmentation results using fully convolutional neural network (FCN) [36], the network architecture of which is shown in Figure 3.2, are shown in Figure 4.2, 4.3 and 4.4.



(a) Slice 1

(b) Slice 2

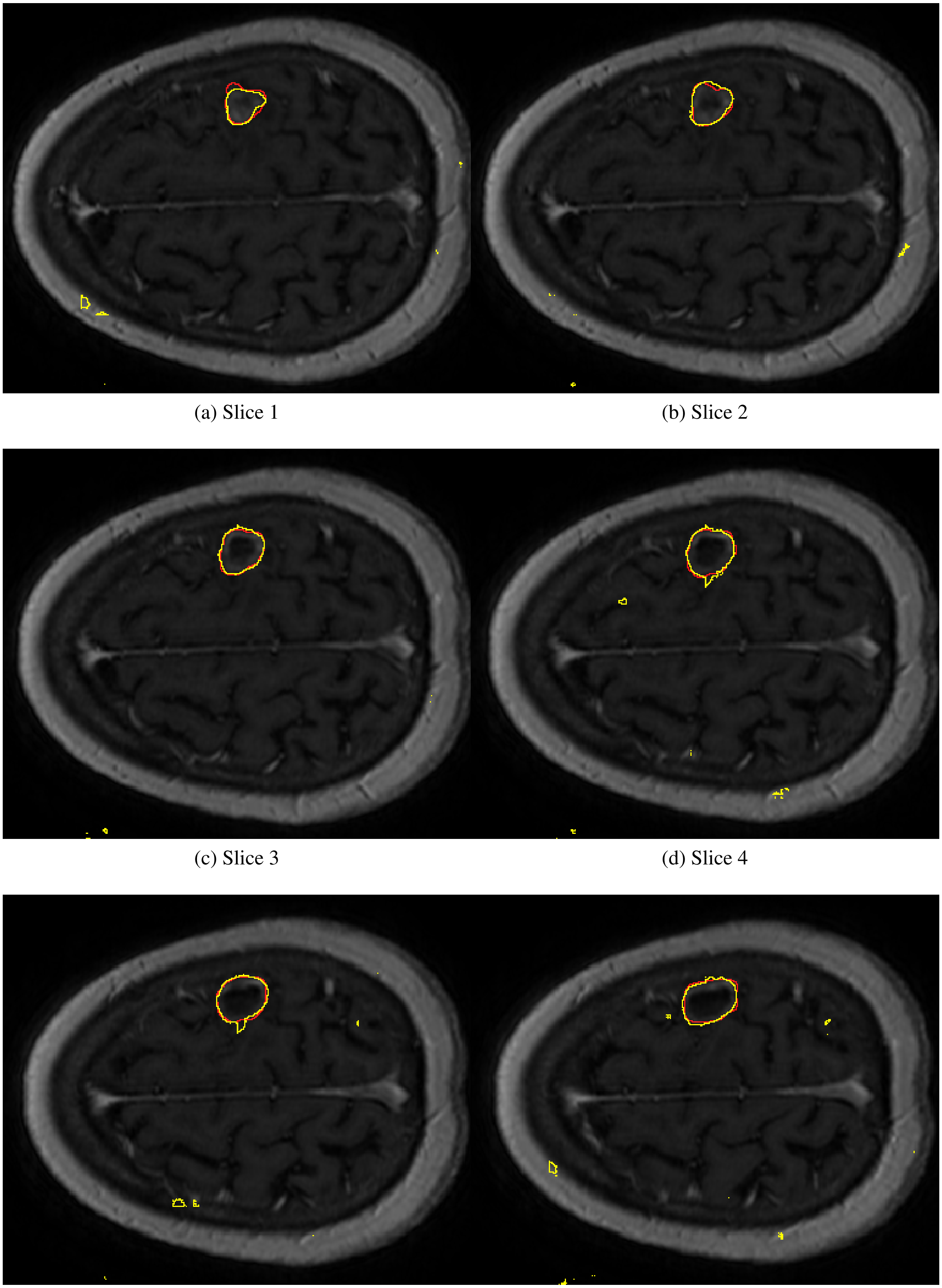
(c) Slice 3

(d) Slice 4

(e) Slice 5

(f) Slice 6

Figure 4.2: Segmentation results of part of one volume using only FCN. The red contour is the boundary of the ground truth, and the yellow contour is the boundary of the segmentation.



(a) Slice 1

(b) Slice 2

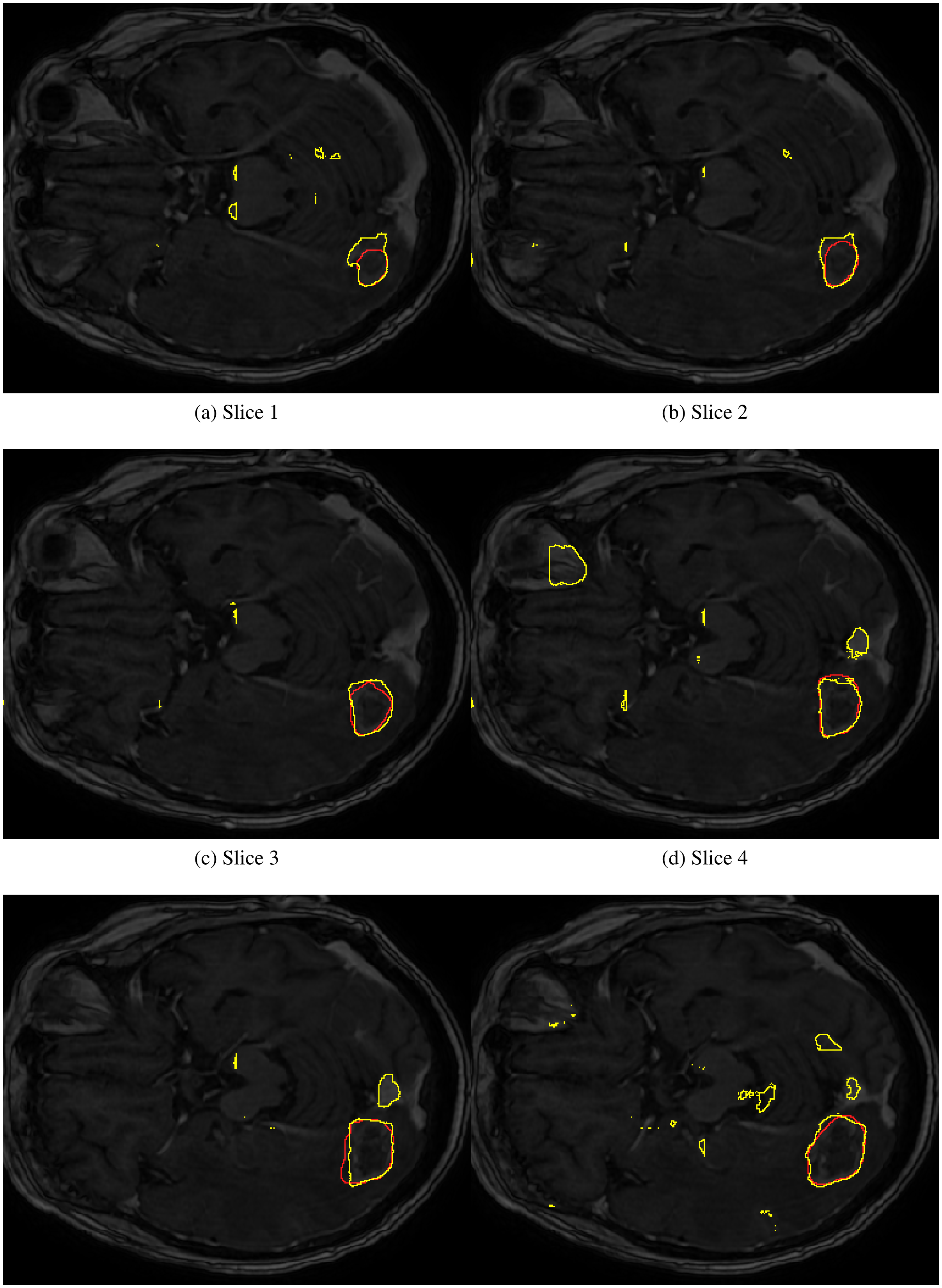
(c) Slice 3

(d) Slice 4

(e) Slice 5

(f) Slice 6

Figure 4.3: Segmentation results of part of one volume using only FCN. The red contour is the boundary of the ground truth, and the yellow contour is the boundary of the segmentation.



(a) Slice 1

(b) Slice 2

(c) Slice 3

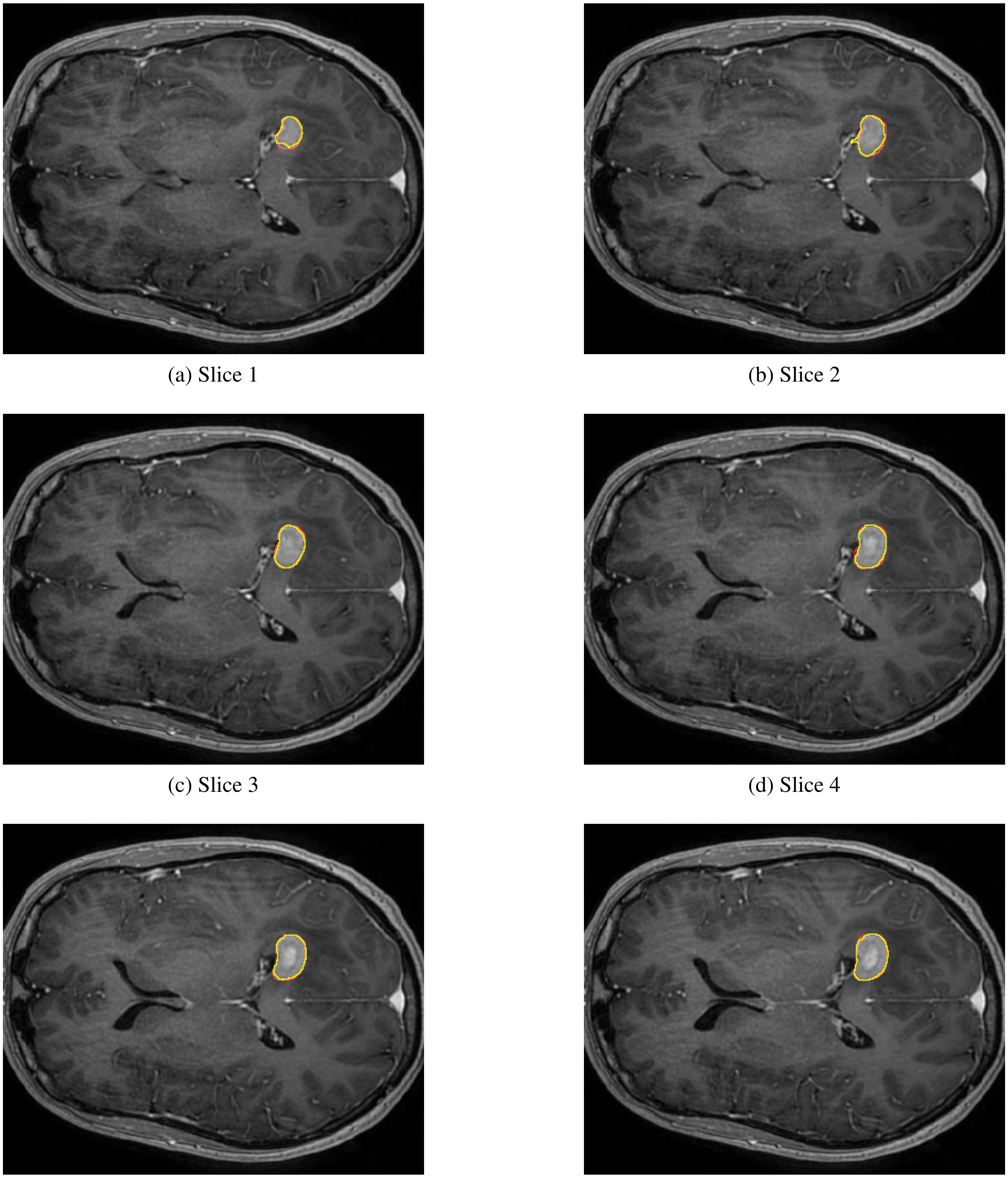
(d) Slice 4

(e) Slice 5

(f) Slice 6

Figure 4.4: Segmentation results of part of one volume using only FCN. The red contour is the boundary of the ground truth, and the yellow contour is the boundary of the segmentation.

U-Net The segmentation results using U-Net [46], the network architecture of which is shown in Figure 3.3, are shown in Figure 4.5, 4.6 and 4.7.



(a) Slice 1

(b) Slice 2

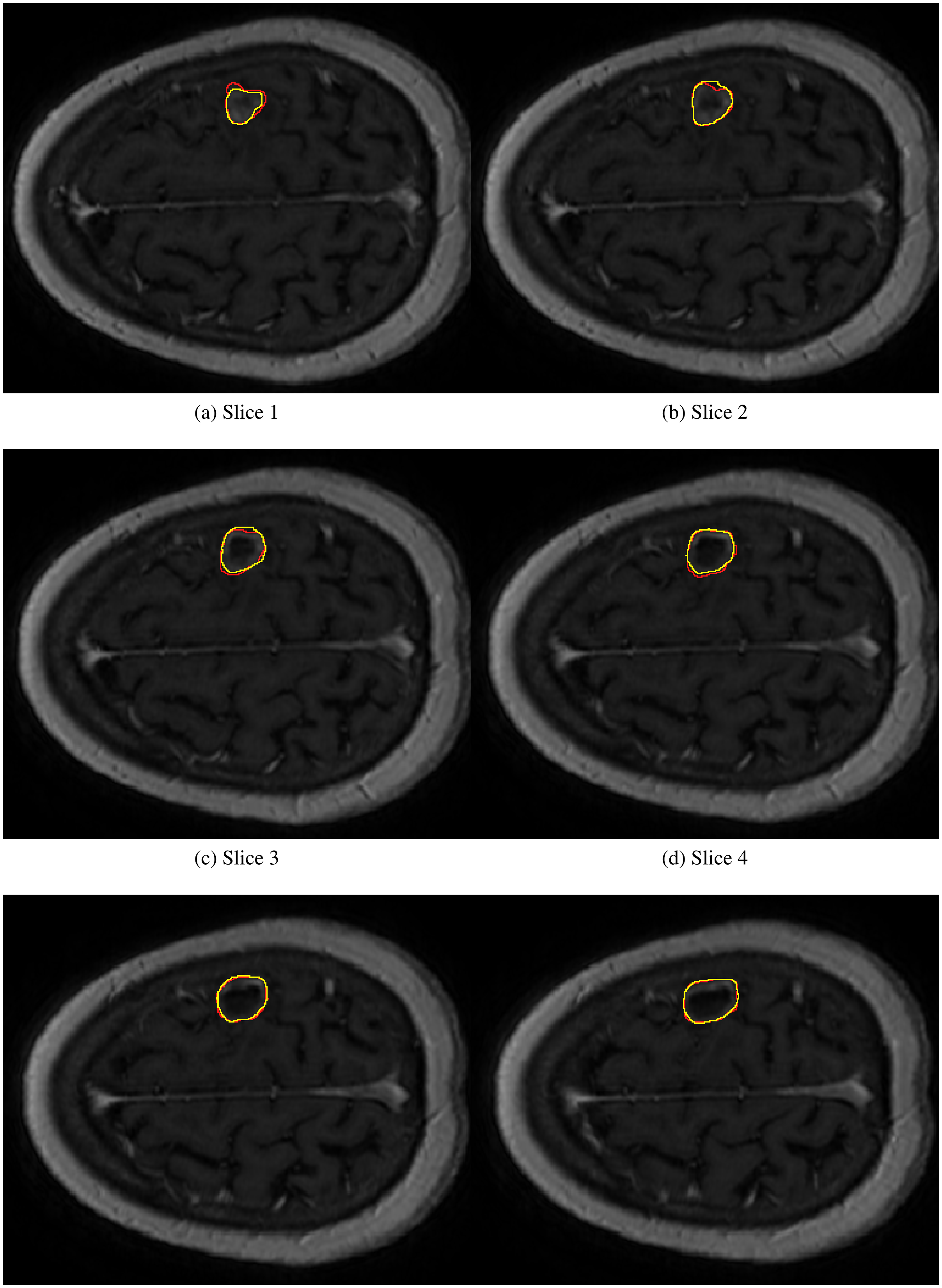
(c) Slice 3

(d) Slice 4

(e) Slice 5

(f) Slice 6

Figure 4.5: Segmentation results of part of one volume using only U-Net. The red contour is the boundary of ground truth, and the yellow contour is the boundary of the segmentation.


(a) Slice 1

(b) Slice 2

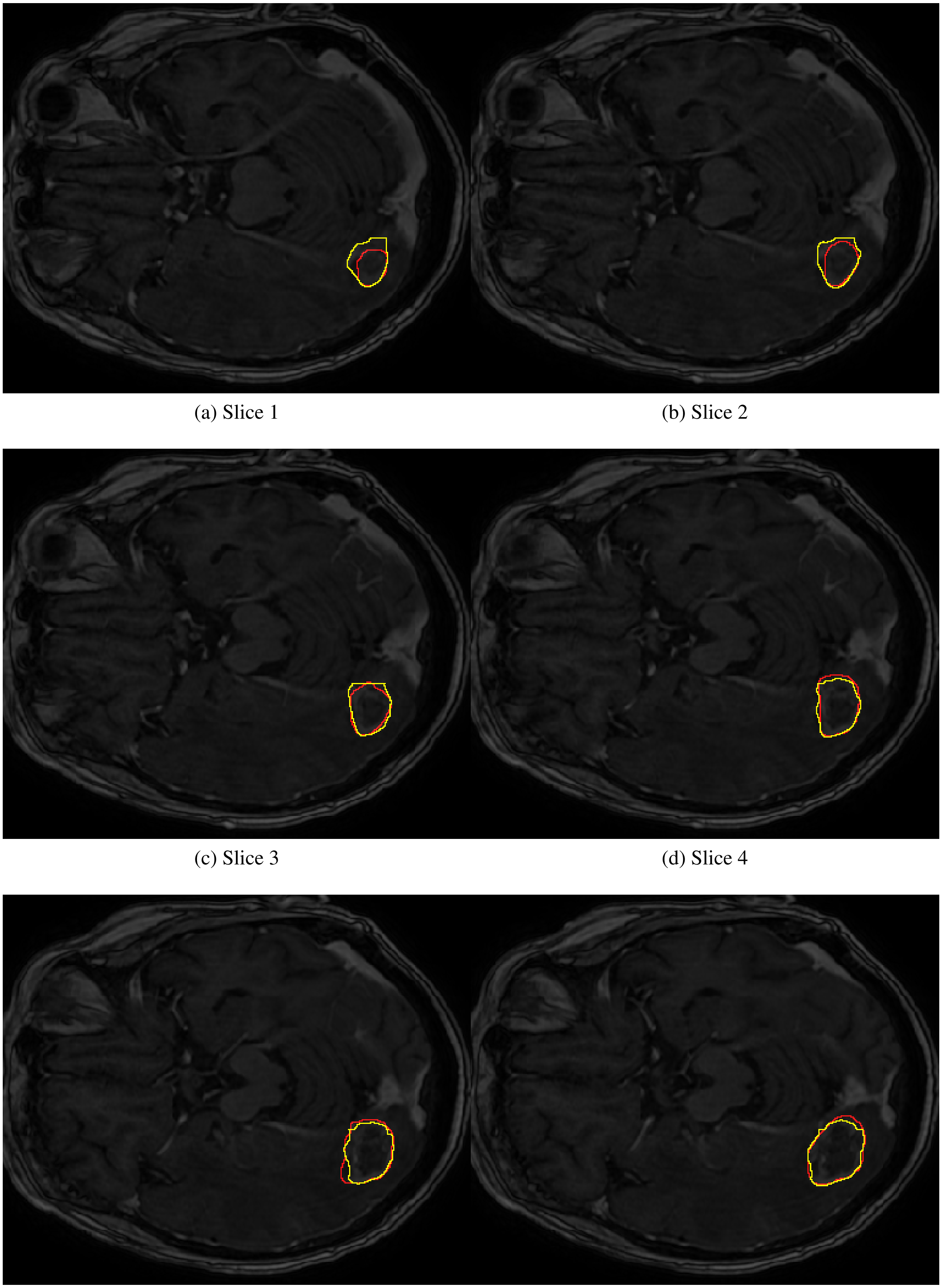
(c) Slice 3

(d) Slice 4

(e) Slice 5

(f) Slice 6

Figure 4.6: Segmentation results of part of one volume using only U-Net. The red contour is the boundary of ground truth, and the yellow contour is the boundary of the segmentation.



(a) Slice 1

(b) Slice 2

(c) Slice 3

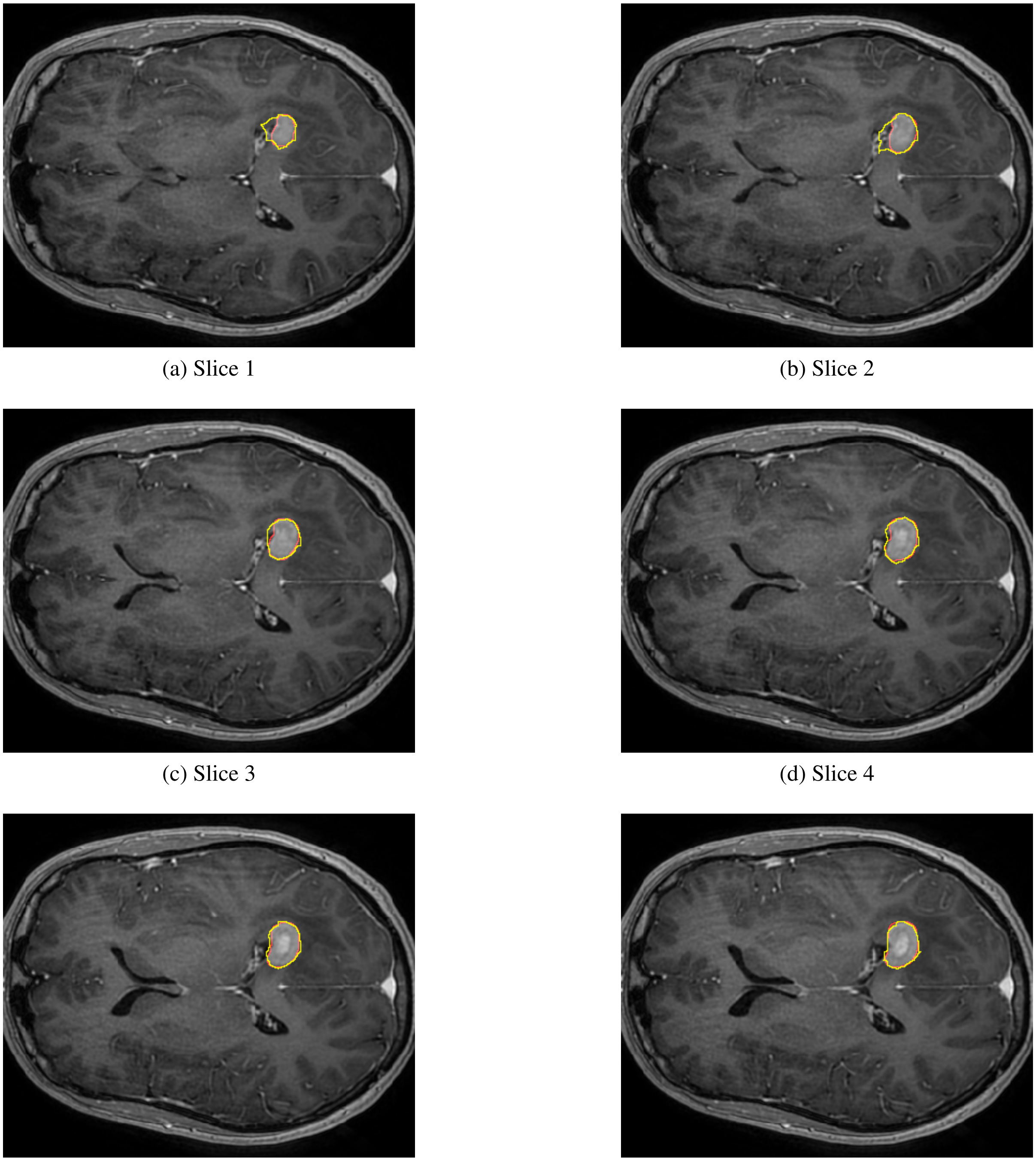
(d) Slice 4

(e) Slice 5

(f) Slice 6

Figure 4.7: Segmentation results of part of one volume using only U-Net. The red contour is the boundary of ground truth, and the yellow contour is the boundary of the segmentation.

Dilated Residual Network The segmentation results using Dilated Residual Network (DRN) [58], the network architecture of which is shown in Figure 3.5, are shown in Figure 4.8, 4.9 and 4.10.



(a) Slice 1

(b) Slice 2

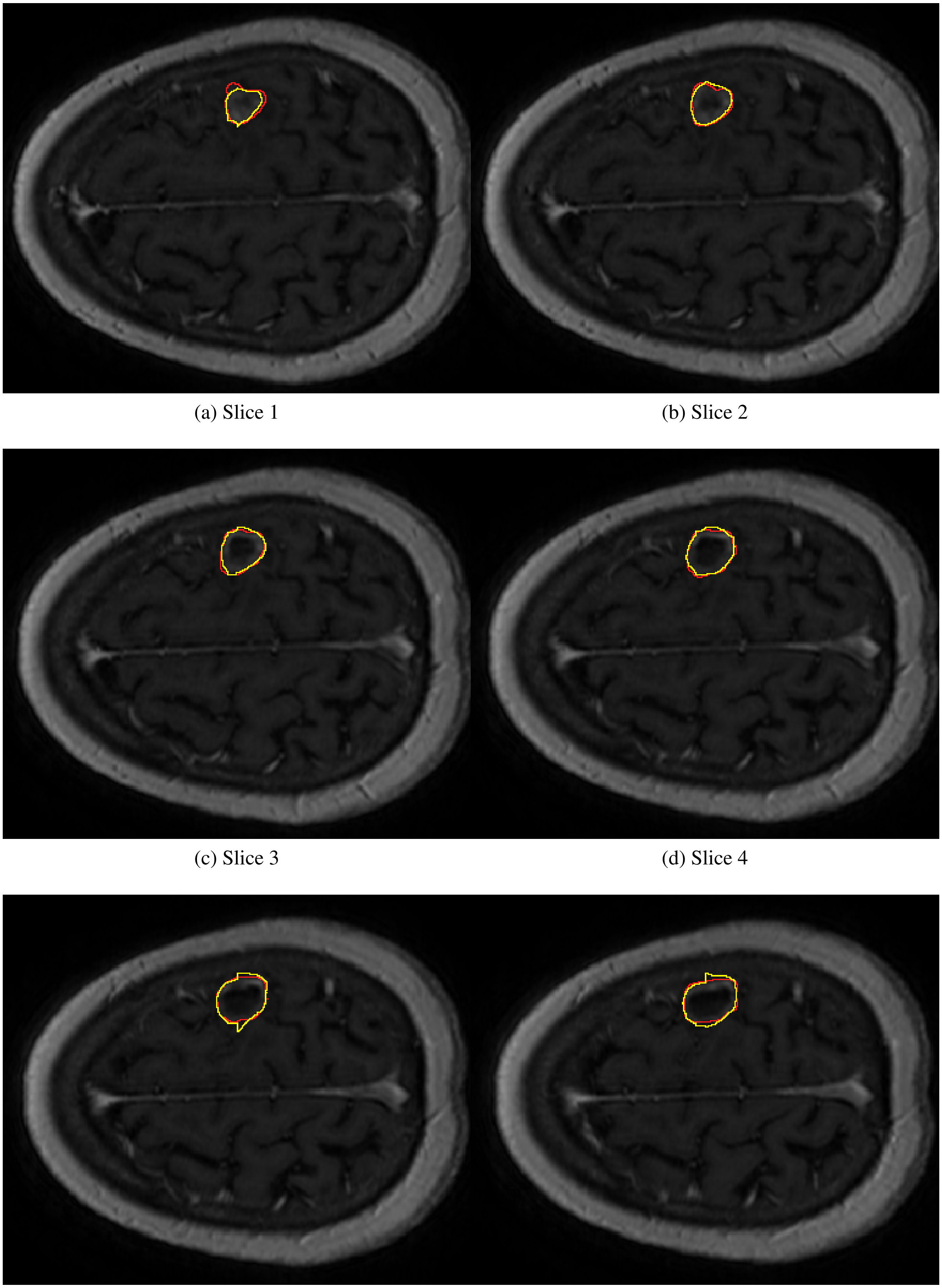
(c) Slice 3

(d) Slice 4

(e) Slice 5

(f) Slice 6

Figure 4.8: Segmentation results of part of one volume using only DRN. The red contour is the boundary of ground truth, and the yellow contour is the boundary of the segmentation.



(a) Slice 1

(b) Slice 2

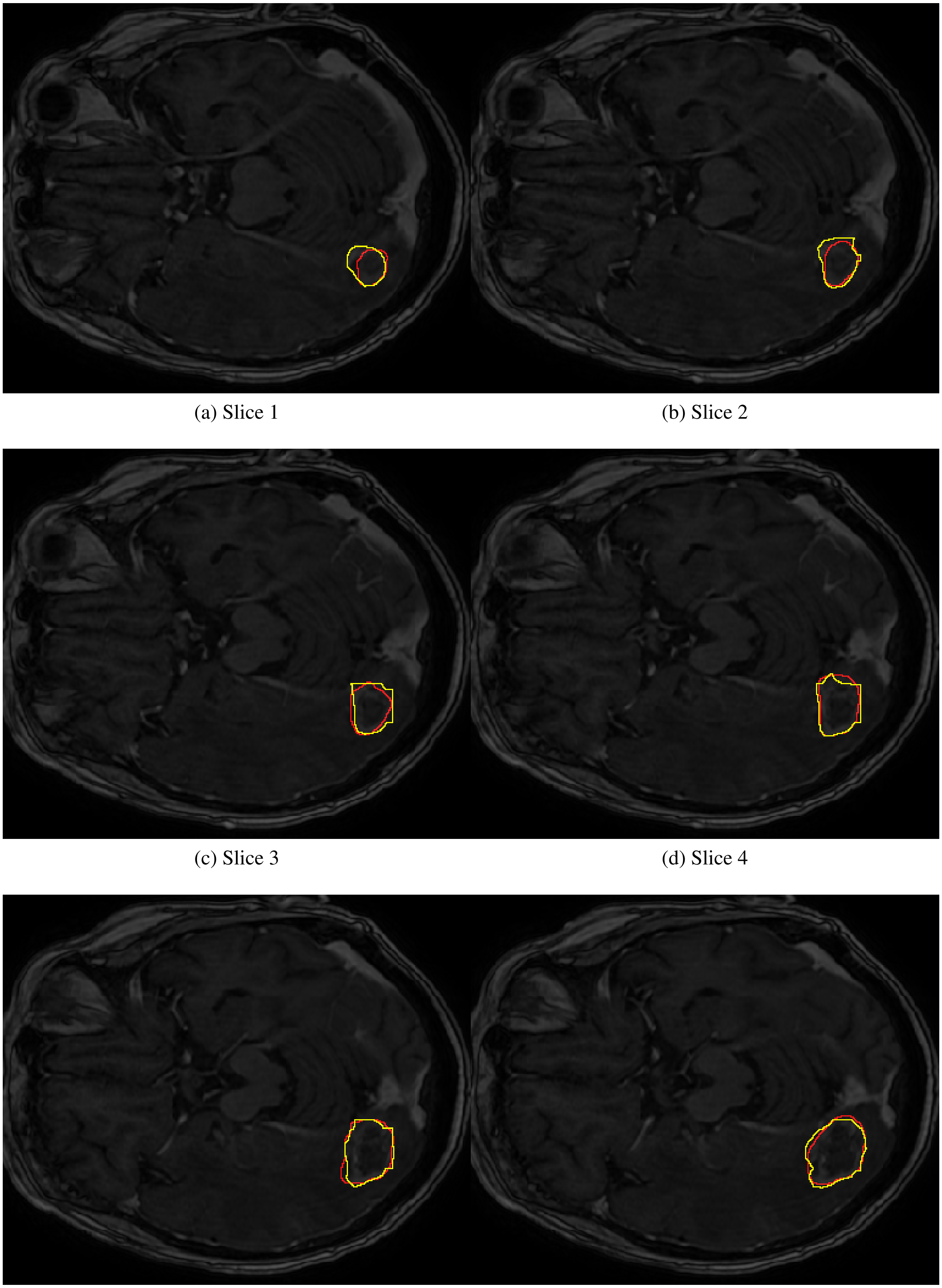
(c) Slice 3

(d) Slice 4

(e) Slice 5

(f) Slice 6

Figure 4.9: Segmentation results of part of one volume using only DRN. The red contour is the boundary of ground truth, and the yellow contour is the boundary of the segmentation.



(a) Slice 1

(b) Slice 2

(c) Slice 3

(d) Slice 4

(e) Slice 5

(f) Slice 6

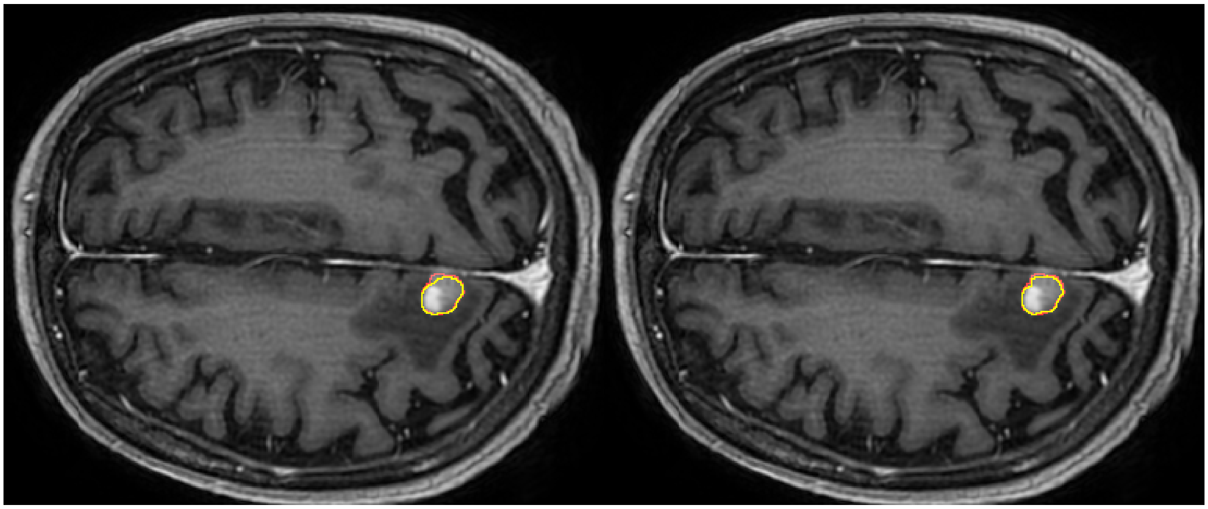
Figure 4.10: Segmentation results of part of one volume using only DRN. The red contour is the boundary of ground truth, and the yellow contour is the boundary of the segmentation.

4.5 Results Comparison of Single vs. Three Slices CNNs

In the following sections, our experiments are all based on U-Net for consistency. In this section, we compare the segmentation results by training, validating and testing on only one slice and three slices. For training and validating on three slices, we take the previous, current and next slice and concatenate them together as images with three channels. We use the ground truth of the middle slice as the ground truth. Figures 4.11, 4.12 and 4.13 show the segmentation results by training on single slice and three slices. The evaluation metrics for the performance of training on single slice and three slices are shown in Table 4.2. As this table illustrates, training on three slices achieves a higher accuracy.

Number of slices	Pixel accuracy	IoU
single slice	0.9973	0.7286
3 slices	0.9976	0.7530

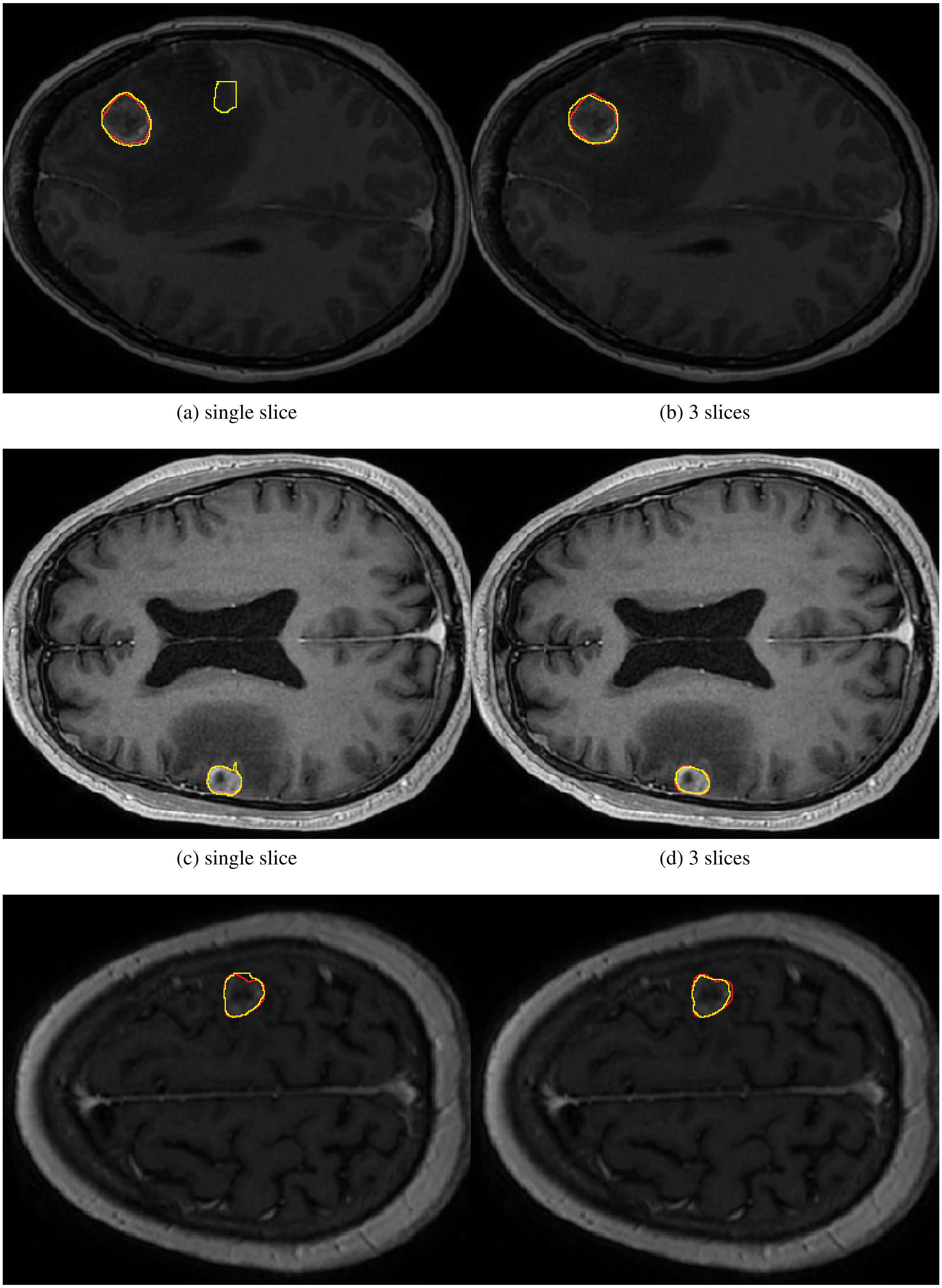
Table 4.2: Evaluation metrics for the performance of training on single and three slices



(a) single slice

(b) 3 slices

Figure 4.11: Comparison of segmentation results using only one slice and 3 slices. For each row, the left image is the segmentation result using only one slice and the right image is the segmentation result using 3 slices. The red contour is the ground truth boundary, and the yellow contour is the segmentation boundary by U-Net.



(a) single slice

(b) 3 slices

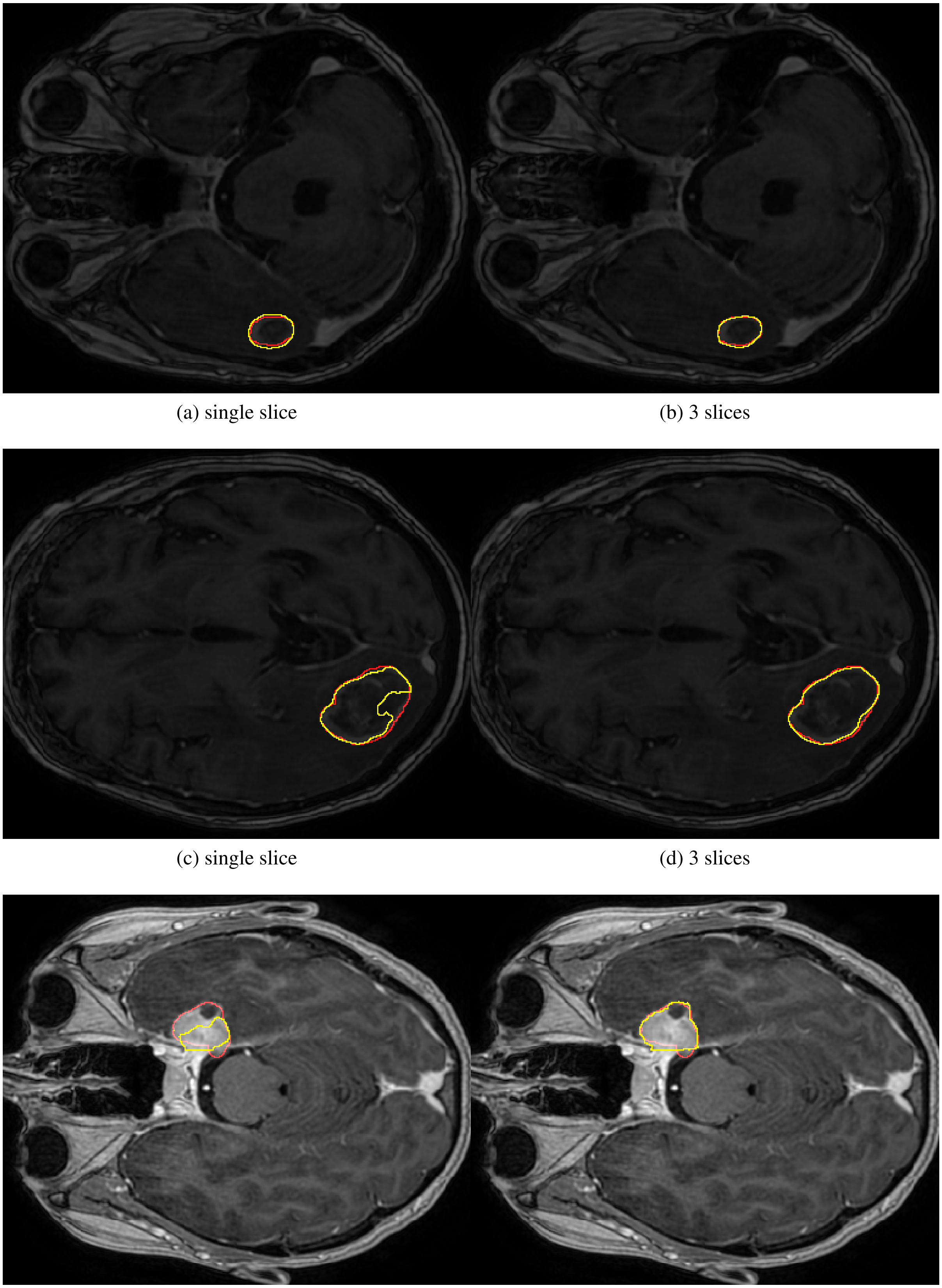
(c) single slice

(d) 3 slices

(e) single slice

(f) 3 slices

Figure 4.12: Comparison of segmentation results using only one slice and 3 slices. For each row, the left image is the segmentation result using only one slice and the right image is the segmentation result using 3 slices. The red contour is the ground truth boundary, and the yellow contour is the segmentation boundary by U-Net.



(a) single slice

(b) 3 slices

(c) single slice

(d) 3 slices

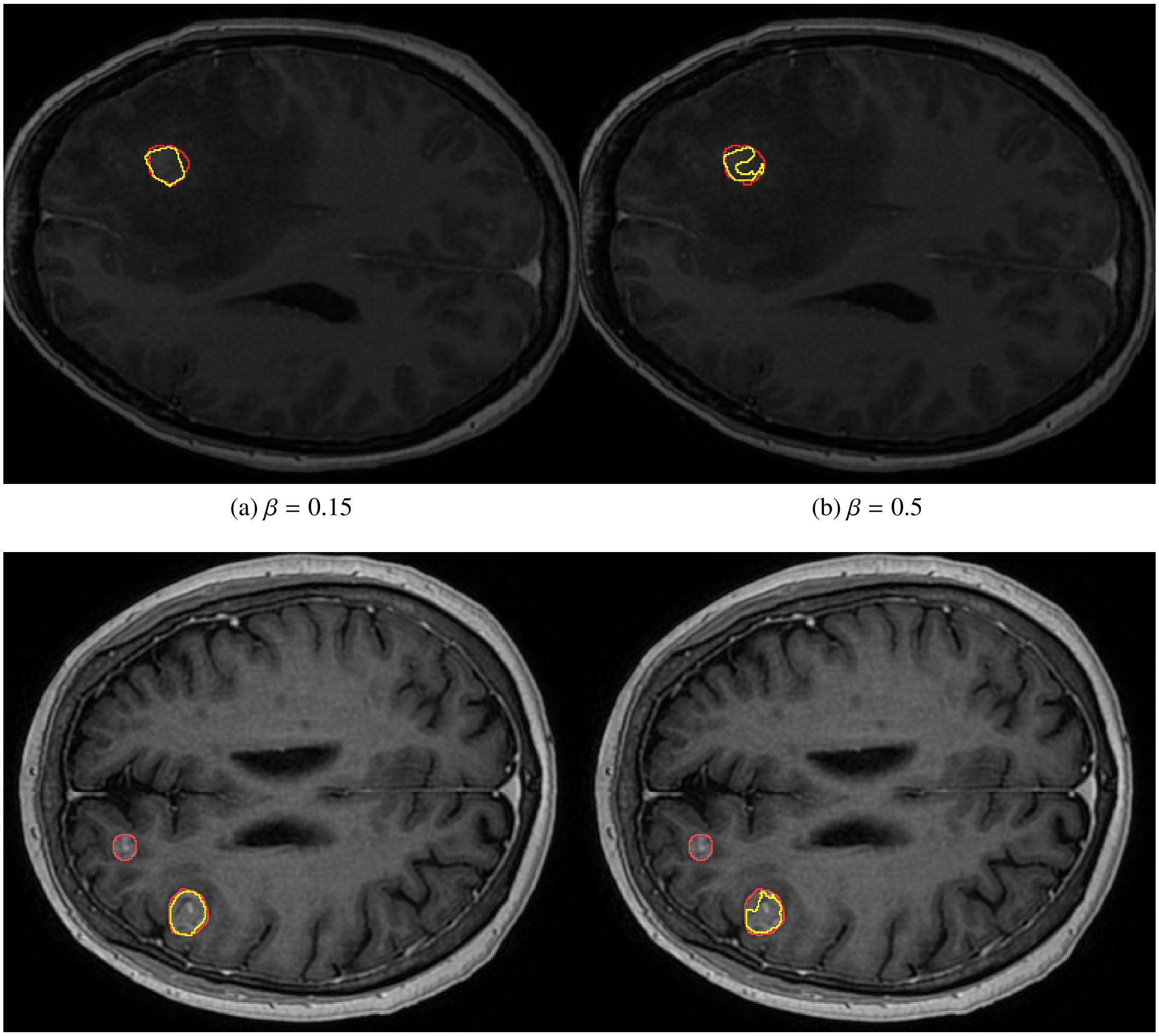
(e) single slice

(f) 3 slices

Figure 4.13: Comparison of segmentation results using only one slice and 3 slices. For each row, the left image is the segmentation result using only one slice and the right image is the segmentation result using 3 slices. The red contour is the ground truth boundary, and the yellow contour is the segmentation boundary by U-Net.

4.6 The effects of different balance weight for loss function

In this section, we compare the segmentation results by using balanced loss function, as in Section 3.2.6, and investigate the effects of the balance weight on the segmentation results. The parameter β in Equation 3.1 controls the relative importance between background and tumor pixels. Table 4.3 shows the effects of balance weight β on the segmentation performance. Figures 4.14, 4.15 and 4.16 show some segmentation results when β = 0.15 and β = 0.5. The performance, according to IoU measure is the best for β = 0.15.



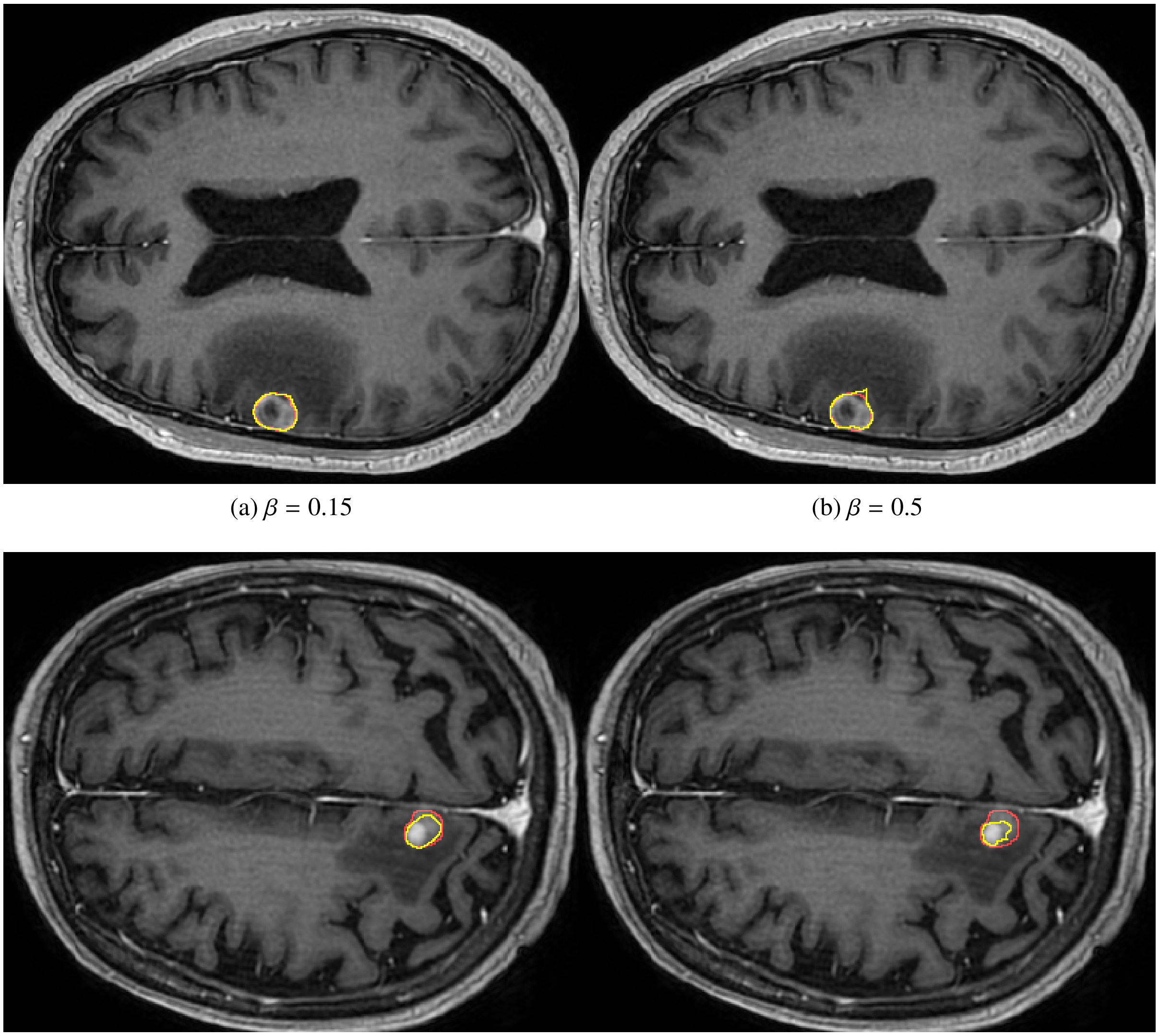
(a) β = 0.15

(b) β = 0.5

(c) β = 0.15

(d) β = 0.5

Figure 4.14: Comparison of segmentation results using balanced loss function when β=0.15 and β=0.5. For each row, the left image is the segmentation result using β=0.15 and the right image is the segmentation result using β=0.5. The red contour is the ground truth boundary, and the yellow contour is the segmentation boundary by U-Net.



(a) β = 0.15

(b) β = 0.5

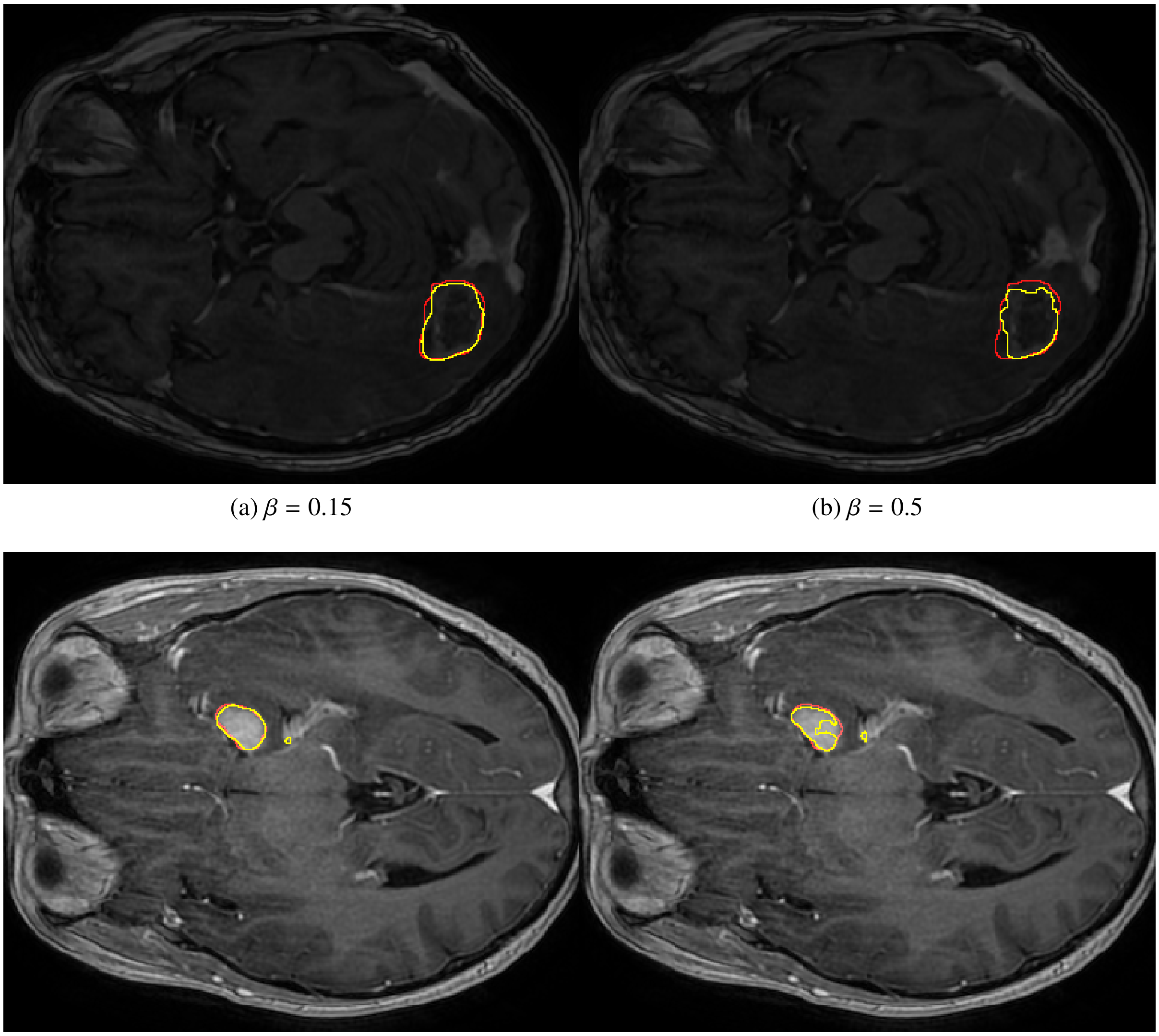
(c) β = 0.15

(d) β = 0.5

Figure 4.15: Comparison of segmentation results using balanced loss function when β=0.15 and β=0.5. For each row, the left image is the segmentation result using β=0.15 and the right image is the segmentation result using β=0.5. The red contour is the ground truth boundary, and the yellow contour is the segmentation boundary by U-Net.

β	Pixel accuracy	IoU
0.01	0.9962	0.6593
0.05	0.9968	0.6877
0.1	0.9976	0.7203
0.15	0.9977	0.7713
0.2	0.9977	0.7558
0.25	0.9975	0.7425
0.3	0.9975	0.7276
0.5	0.9976	0.7530

Table 4.3: Evaluation metrics with different balance weight for the balanced loss function



(a) β = 0.15

(b) β = 0.5

(c) β = 0.15

(d) β = 0.5

Figure 4.16: Comparison of segmentation results using balanced loss function when β=0.15 and β=0.5. For each row, the left image is the segmentation result using β=0.15 and the right image is the segmentation result using β=0.5. The red contour is the ground truth boundary, and the yellow contour is the segmentation boundary by U-Net.

4.7 Segmentation comparison by only CNNs and combining CNNs with graph cut regularization

We tried different combinations of U-Net with different data terms, presented in Section 3.3.1, for graph cut energy minimization. The U-Net are trained with balanced loss function when $\beta = 0.15$ and using three slices. Table 4.4 shows the performance of different combination. Table 4.5 shows the performance of different threshold for the weighted negative log likelihood data term. We observe that graph cut optimization with every choice of data terms improves over the results of just using U-Net.

Data term	IoU
without graph cut	0.7713
(a) hard constraint for probability	0.7788
(b) hard constraint for log likelihood	0.7798
(c) 0 and 1 cost	0.8078
(d) negative log likelihood	0.7914
(e) negative log likelihood and distance transform	0.8033
(f) weighted negative log likelihood	0.8041

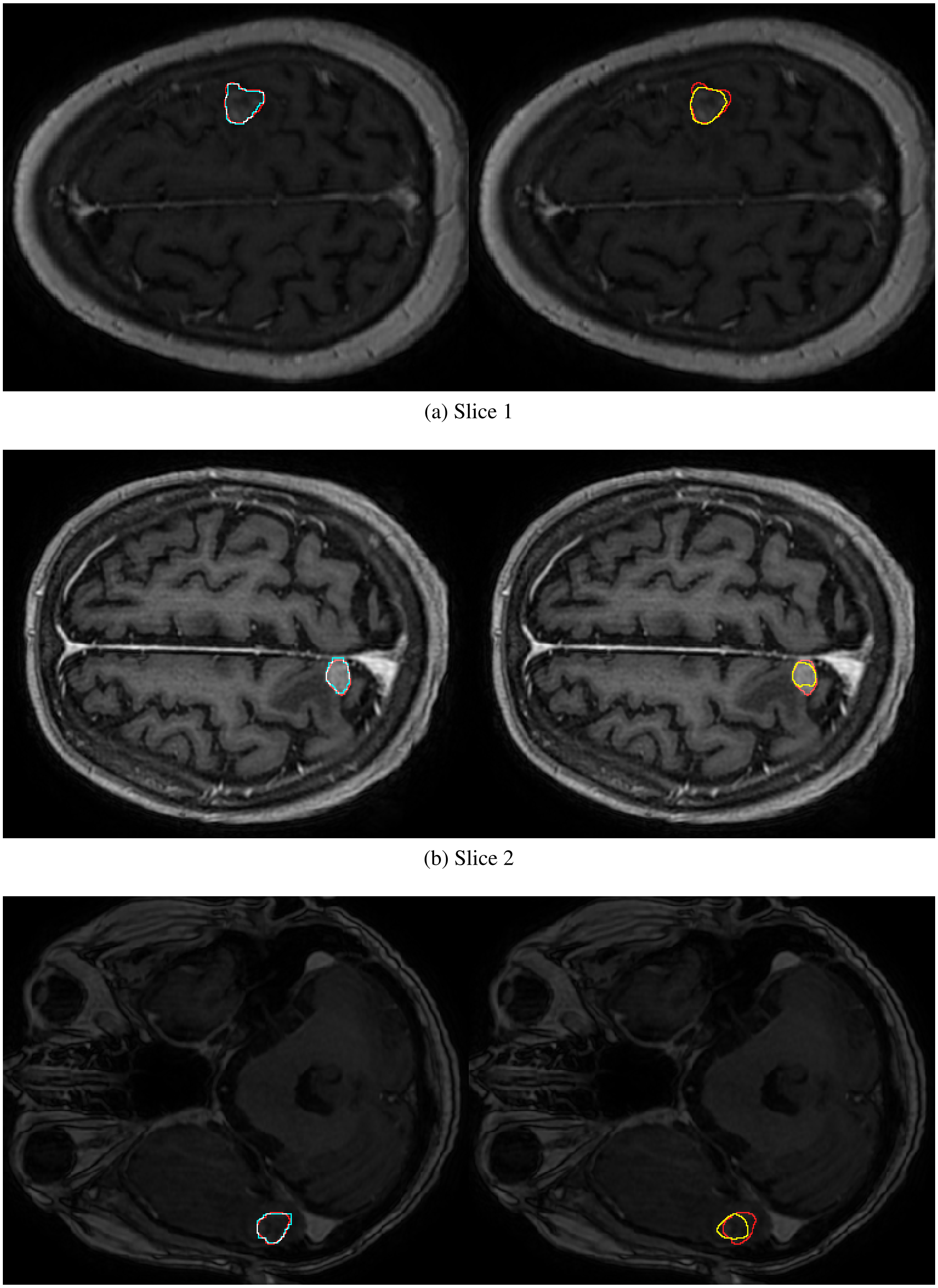
Table 4.4: IoU of combining U-Net with graph cut regularization.

threshold	5	10	20	24	25	26	30	40
IoU	0.7975	0.8019	0.8037	0.8038	0.8041	0.8038	0.8040	0.8036

Table 4.5: Performance evaluation by setting different threshold for data term f.

4.7.1 Successful cases of improving the segmentation results

The successful improvement cases of segmentation are shown in Figure 4.17 and Figure 4.18.

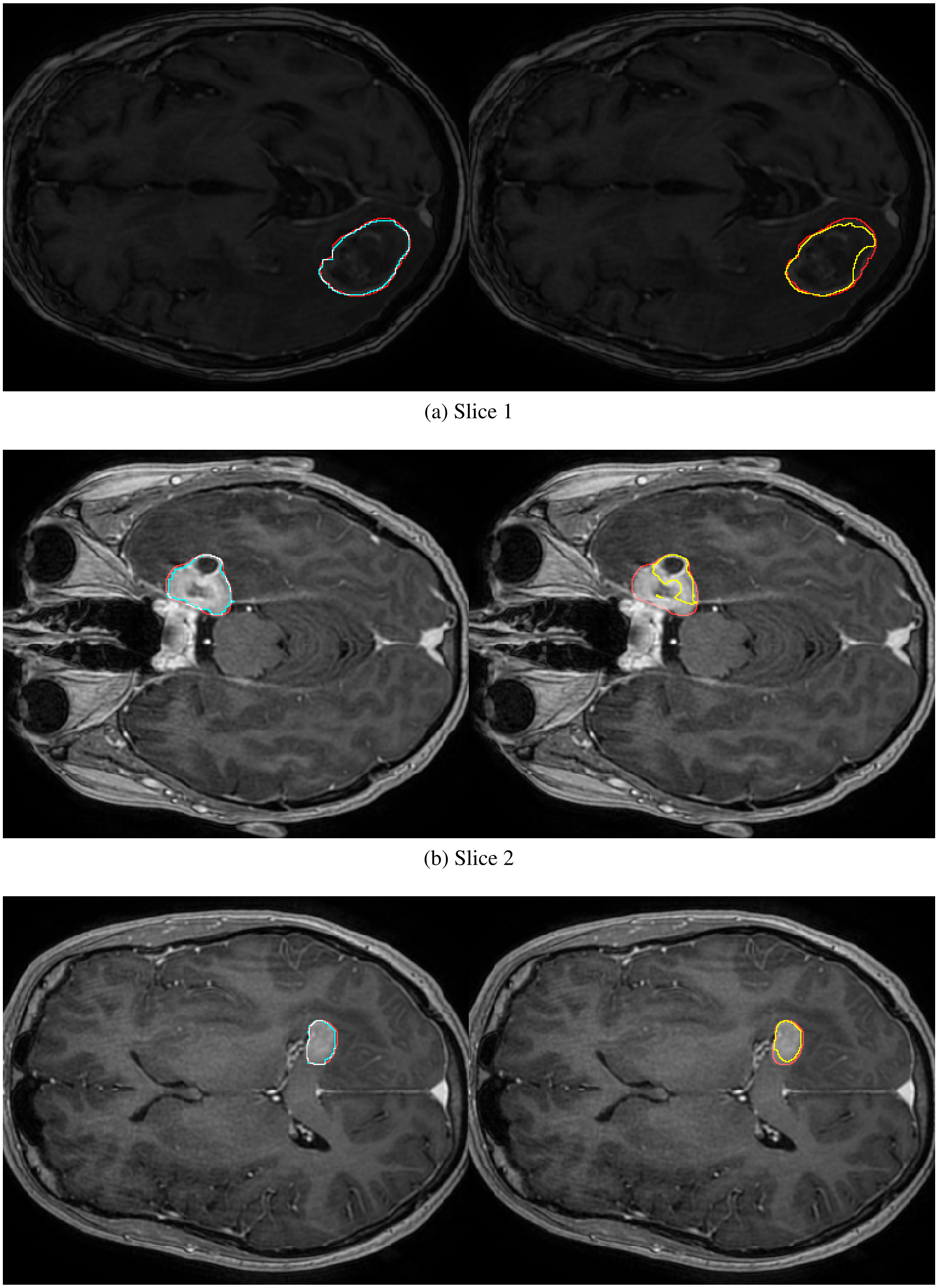


(a) Slice 1

(b) Slice 2

(c) Slice 3

Figure 4.17: Comparison of segmentation results using only U-Net and combination of U-Net with graph cut regularization. For each row, the right image is the segmentation using only U-Net and the left image is the segmentation using U-Net and graph cut regularization. The red contour is the ground truth boundary, the blue contour is the segmentation boundary by combing U-Net with graph cut regularization, and the yellow contour is the segmentation boundary by U-Net.



(a) Slice 1

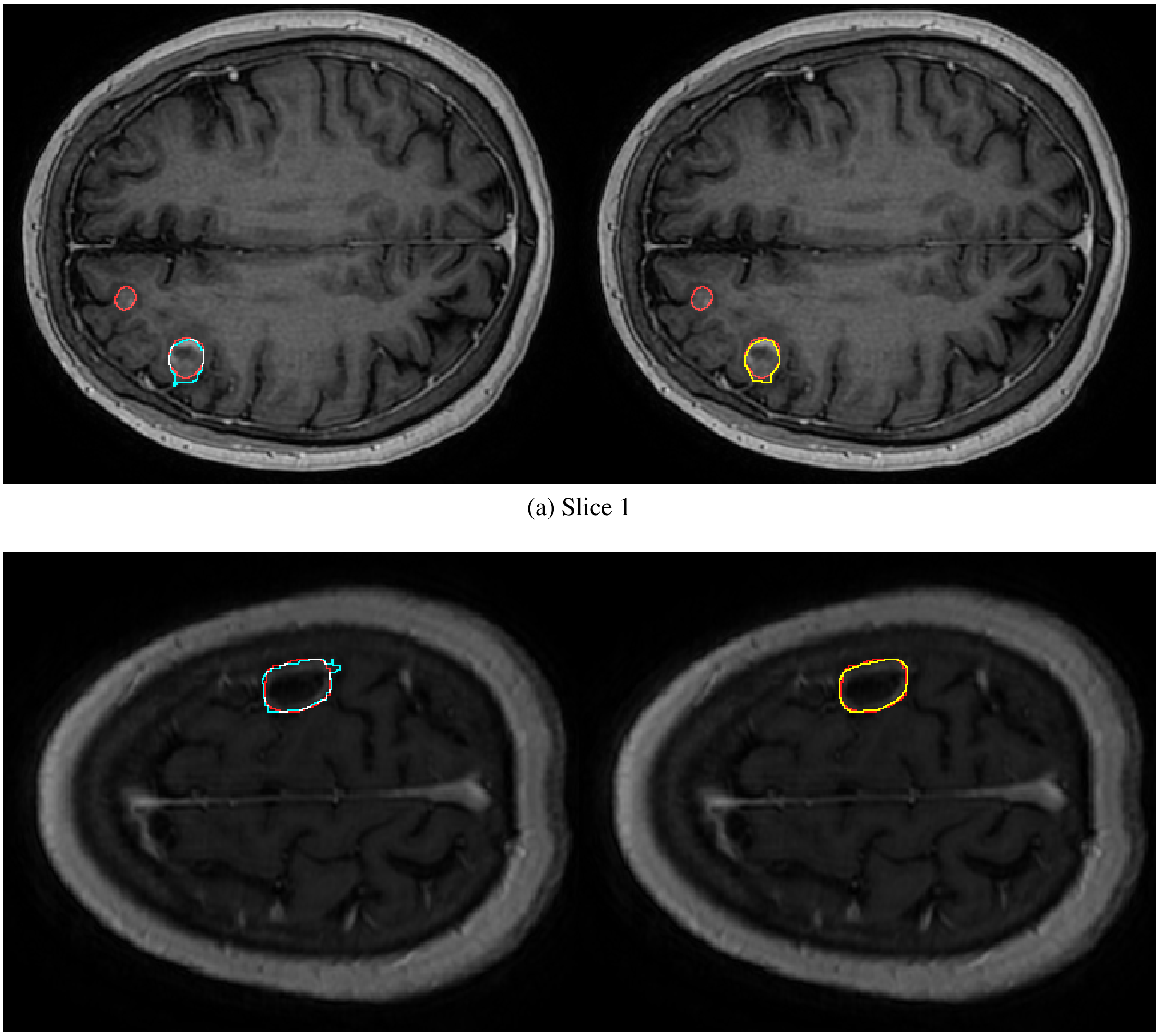
(b) Slice 2

(c) Slice 3

Figure 4.18: Comparison of segmentation results using only U-Net and combination of U-Net with graph cut regularization. For each row, the right image is the segmentation result using only U-Net and the left image is the segmentation result using U-Net and graph cut regularization. The red contour is the ground truth boundary, the blue contour is the segmentation boundary by combing U-Net with graph cut regularization, and the yellow contour is the segmentation boundary by U-Net.

4.7.2 Decrease in accuracy when using graph cuts

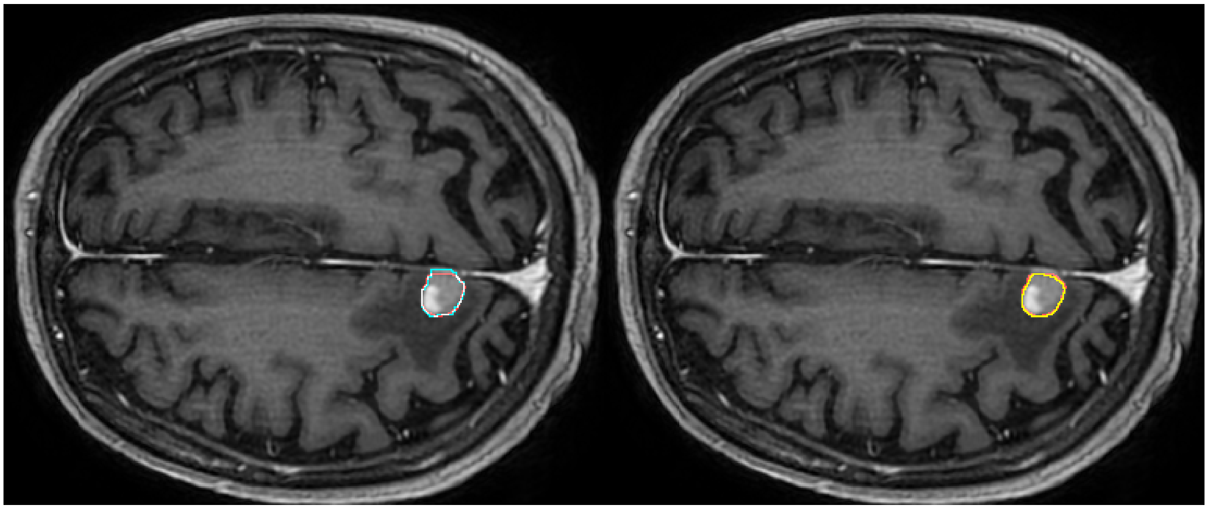
The combination of U-Net with graph cut regularization does not always improve the segmentation results and the worse cases of segmentation results are shown in Figure 4.19.



(a) Slice 1

(b) Slice 2

Figure 4.19: Comparison of segmentation results using only U-Net and combination of U-Net with graph cut regularization. For each row, the right image is the segmentation result using only U-Net and the left image is the segmentation result using U-Net and graph cut regularization. The red contour is the ground truth boundary, the blue contour is the segmentation boundary by combing U-Net with graph cut regularization, and the yellow contour is the segmentation boundary by U-Net.



(c) Slice 3

Figure 4.19: Comparison of segmentation results using only U-Net and combination of U-Net with graph cut regularization. For each row, the right image is the segmentation result using only U-Net and the left image is the segmentation result using U-Net and graph cut regularization. The red contour is the ground truth boundary, the blue contour is the segmentation boundary by combing U-Net with graph cut regularization, and the yellow contour is the segmentation boundary by U-Net.

4.8 The effects of the crop size on the segmentation results

In this section, we compare the segmentation results by training on smaller crop size with 128×128 and larger crop sizes with 160×160 and 192×192. Table 4.6 shows the evaluation metrics by training on smaller crops and larger crops. When training on crops of size 128×128, we need nearly two days to complete one experiment; when training on crops of larger sizes, we need more time to finish a complete experiment, but the performance improvement is not significant. Since all experiment settings are the same except crop size, to speed up experiment, we use crops of size 128×128 throughout the thesis.

Crop size	Pixel accuracy	IoU
128×128	0.9973	0.7286
160×160	0.9973	0.7292
192×192	0.9974	0.7318

Table 4.6: Evaluation metrics for the performance of training on smaller crops and larger crops

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we proposed an effective automatic brain tumor segmentation approach that combines deep convolutional neural networks with graph cut regularization.

First, we train deep convolutional neural networks on the overlapping crops of whole slices of brain tumor data. Since the tumor pixels account for a very small portion in the whole slice image, segmenting tumors from the background is a highly imbalanced dense prediction task. We used the loss function that takes the imbalance of the training data into consideration. The next step is to combine the results on overlapping crops to obtain segmentation on a whole brain slice. We did this by a simple union operation.

Second, we applied graph cut regularization using the segmentation probability map obtained by deep convolutional neural networks to further improve the deep neural network segmentation results. We explored and evaluated several ways in which the tumor probabilities learned by the neural network can be converted to the data terms required by graph cut regularization.

Experimental results showed that the intersection of union for tumor is improved by combing deep neural network with graph cut regularization compared to only use deep neural network.

5.2 Future Work

In this section, we discuss several possible future research directions.

5.2.1 More robust deep learning model

One of the difficulty in automatic brain tumor segmentation is the similarity between background and tumor, this will cause neural network can not discriminate tumor and background pixels. The neural network will classify some background pixels as tumor pixels, and some tumor pixels as background pixels. Thus, one of the future direction is to develop more robust models to have more powerful discriminate ability to differentiate the tumor and background pixels.

5.2.2 More effective loss function for training deep CNNs

The current loss function for training deep CNNs is the cross entropy loss function and its simple modification, one of the possible direction is to develop more effective loss function that can better focus on the tumor pixels and differentiate them from background pixels.

5.2.3 More effective data term

We have tried different data terms for graph cut segmentation, but the performance of different data terms are comparable. So, we need to develop more effective data terms that can utilize neural network output to further improve the neural network segmentation results.

Bibliography

[1] Varghese Alex, Mohammed Safwan, and Ganapathy Krishnamurthi. Brain tumor segmentation from multi modal mr images using fully convolutional neural network. In *Proceedings of MICCAI workshop on Multimodal Brain Tumor Segmentation Challenge (BRATS)*, 2017.

[2] P. H. A. Amorim, V. S. Chagas, G. G. Escudero, D. D. C. Oliveira, S. M. Pereira, H. M. Santos, and A. A. Scussel. 3d u-nets for brain tumor segmentation in miccai 2017 brats challenge. In *Proceedings of MICCAI workshop on Multimodal Brain Tumor Segmentation Challenge (BRATS)*, 2017.

[3] Stefan Bauer, Lutz-P. Nolte, and Mauricio Reyes. Fully automatic segmentation of brain tumor images using support vector machine classification in combination with hierarchical conditional random field regularization. In *Proceedings of International Conference on Medical Image Computing and ComputerAssisted Intervention (MICCAI)*, pages 354–361, 2011.

[4] Stefan Bauer, Roland Wiest, Lutz-P Nolte, and Mauricio Reyes. A survey of mri based medical image analysis for brain tumor studies. *Physics in medicine and biology*, 58(13):97–129, 2013.

[5] Neil Birkbeck, Dana Cobzas, Martin Jagersand, Albert Murtha, and Tibor Kesztyues. An interactive graph cut method for brain tumor segmentation. In *2009 Workshop on Applications of Computer Vision (WACV)*, 2009.

[6] Christopher Michael Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

[7] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[8] Shilei Cao, Buyue Qian, Changchang Yin, Xiaoyu Li, and Shiyu Chang. 3d u-net for multimodal brain tumor segmentation. In *Proceedings of MICCAI workshop on Multimodal Brain Tumor Segmentation Challenge (BRATS)*, 2017.

[9] Danfeng Chen. *Interative Brain Tumor Segmentation with Inclusion*. Master's thesis, University of Western Ontario, 2016.

[10] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv:1606.00915*, 2016.

[11] Brian M. Dale, Mark A. Brown, and Richard C. Semelka. *MRI: Basic Principles and Applications*. Wiley-Blackwell, 2015.

[12] Simon S. Du, Chi Jin, Jason D. Lee, Michael I. Jordan, Barnabas Poczos, and Aarti Singh. Gradient descent can take exponential time to escape saddle points. *arXiv:1705.10412*, 2017.

[13] Ross Girshick. Fast r-cnn. In *International Conference on Computer Vision (ICCV)*, 2015.

[14] Michael Goetz, Christian Weber, Josiah Bloecher, Bram Stieltjes, Hans-Peter Meinzer, and Klaus Maier-Hein. Extremely randomized trees based brain tumor segmentation. In *Proceedings of BRATS Challenge - Medical Image Computing and ComputerAssisted Intervention (MICCAI)*, 2014.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[16] Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, and Hugo Larochelle. Brain tumor segmentation with deep neural networks. *Medical Image Analysis*, 35:18–31, 2017.

[17] Mohammad Havaei, Pierre-Marc Jodoin, and Hugo Larochelle. Efficient interactive brain tumor segmentation as within-brain knn classification. In *Proceedings of 22nd International Conference on Pattern Recognition (ICPR)*, Stockholm, Sweden, 2014.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, USA, 2016.

[19] Tsachi Hershkovich, Tamar Shalmon, Ohad Shitrit, Nir Halay, Bjoern H. Menze, Irit Dolgopyat, Itamar Kahn, Ilan Shelef, and Tammy Riklin Raviv. Probabilistic model for 3d

interactive segmentation. *Journal of Computer Vision and Image Understanding*, 151:47–60, 2016.

[20] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurons in the cats striate cortex. *Journal of Physiology*, 148:574–591, 1959.

[21] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction, and functional architecture in the cats visual cortex. *Journal of Physiology*, 160:106–154, 1962.

[22] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology*, 195:215–243, 1968.

[23] zgn iek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation. 2016.

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of 32nd International Conference on Machine Learning (ICML)*, Lille, France, 2015.

[25] Mobarakol Islam and Hongliang Ren. Fully convolutional network with hypercolumn features for brain tumor segmentation. In *Proceedings of MICCAI workshop on Multimodal Brain Tumor Segmentation Challenge (BRATS)*, 2017.

[26] Yangqing Jia. Caffe solver. `http://caffe.berkeleyvision.org/tutorial/solver.html`.

[27] Lester Randolph Ford Jr and Delbert Ray Fulkerson. *Flows in networks*. Princeton university press, 2015.

[28] Andrej Karpathy. Cs231n: Convolutional neural networks for visual recognition. `http://cs231n.github.io/transfer-learning/`.

[29] Ron Kikinis and Steve Pieper. 3d slicer as a tool for interactive brain tumor segmentation. In *2011 IEEE Annual International Conference on Engineering in Medicine and Biology Society (EMBC)*, Boston, USA, 2011.

[30] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations (ICLR)*, San Diego, USA, 2015.

[31] Vladimir Kolmogorov and Ramin Zabin. What energy functions can be minimized via graph cuts? *IEEE transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159, 2004.

[32] Yan LeCun. Generalization and network design strategies. Technical report, University of Toronto, 1989.

[33] Jason D. Lee, Ioannis Panageas, Georgios Piliouras, Max Simchowitz, Michael I. Jordan, and Benjamin Recht. First-order methods almost always avoid saddle points. *arXiv:1710.07406*, 2017.

[34] Xiaogang Li, Xiang Zhang, and Zhigang Luo. Brain tumor segmentation via 3d fully dilated convolutional networks. In *Proceedings of MICCAI workshop on Multimodal Brain Tumor Segmentation Challenge (BRATS)*, 2017.

[35] Liqun Liu. *Brain Tumor Segmentation with Minimal User Assitance*. Master's thesis, University of Western Ontario, 2014.

[36] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, USA, 2015.

[37] Marc Moreno Lopez and Jonathan Ventura. Dilated convolutions for brain tumor segmentation in mri scans. In *Proceedings of MICCAI workshop on Multimodal Brain Tumor Segmentation Challenge (BRATS)*, 2017.

[38] Holschneider M, Kronland-Martinet R, Morlet J, and Tchamitchian P. A real-time algorithm for signal analysis with the help of the wavelet transform. *Wavelets: Time-Frequency Methods and Phase Space*, pages 286–297, 1989.

[39] Ofer Matan, Christopher J.C. Burges, Yann Le Cun, and John S. Denker. Multi-digit recognition using a space displacement neural network. In *Advances in Neural Information Processing Systems (NIPS)*, 1991.

[40] Richard McKinley, Alain Jungo, Roland Wiest, and Mauricio Reyes. Pooling-free fully convolutional networks with dense skip connections for semantic segmentation, with application to brain tumor segmentation. In *Proceedings of MICCAI workshop on Multimodal Brain Tumor Segmentation Challenge (BRATS)*, 2017.

[41] Raphael Meier, Stefan Bauer, Johannes Slotboom, and Mauricio Reyes. Appearance and context-sensitive features for brain tumor segmentation. In *Proceedings of BRATS Challenge - Medical Image Computing and ComputerAssisted Intervention (MICCAI)*, 2014.

[42] Bjoern H. Menze, Andras Jakab, Stefan Bauer, and etc. The multimodal brain tumor image segmentation benchmark (brats). *IEEE Transactions on Medical Imaging*, 34(10):1993–2004, 2015.

[43] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27:372376, 1983.

[44] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, 2015.

[45] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[46] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proceedings of the 18th International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, Munich, Germany, 2015.

[47] Mazhar Shaikh, Gagan Acharya, Ganesh Anand, Abhijit Amrutkar, Varghese Alex, and Ganapathy Krishnamurthi. Brain tumor segmentation using dense fully convolutional neural network. In *Proceedings of MICCAI workshop on Multimodal Brain Tumor Segmentation Challenge (BRATS)*, 2017.

[48] Mark J Shensa. The discrete wavelet transform: wedding the a trous and mallat algorithms. *IEEE Transactions on Signal Processing*, 40(10):2464–2482, 1992.

[49] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of International Conference on Learning Representations (ICLR)*, San Diego, USA, 2015.

[50] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[51] Nagesh K. Subbanna, Doina Precup, D. Louis Collins, and Tal Arbel. Hierarchical probabilistic gabor and mrf segmentation of brain tumours in mri volumes. In *Proceedings of BRATS Challenge - Medical Image Computing and ComputerAssisted Intervention (MICCAI)*, pages 751–758. Springer, 2013.

[52] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of 32nd International Conference on Machine Learning (ICML)*, Georgia, USA, 2013.

[53] G Urban, M Bendszus, F Hamprecht, and J Kleesiek. Multi-modal brain tumor segmentation using deep convolutional neural networks. In *Proceedings of MICCAI workshop on Multimodal Brain Tumor Segmentation Challenge (BRATS)*, 2014.

[54] Olga Veksler. *Efficient graph-based energy minimization methods in computer vision*. PhD thesis, Cornell University, 1999.

[55] Shijun Wang and Ronald M Summers. Machine learning and radiology. *Medical image analysis*, 16(5):933–951, 2012.

[56] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *Proceedings of IEEE International Conference on Computer Vision*, 2015.

[57] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *Proceedings of 2016 International Conference on Learning Representations (ICLR)*, Caribe Hilton, San Juan, Puerto Rico, 2016.

[58] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. In *Proceedings of 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Hawaii, USA, 2017.

[59] Jure Zbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 17:1–32, 2016.

[60] Jin Zhu, Duo Wang, Zhongzhao Teng, and Pietro Lio. A multi-pathway 3d dilated convolutional neural network for brain tumor segmentation. In *Proceedings of MICCAI workshop on Multimodal Brain Tumor Segmentation Challenge (BRATS)*, 2017.

[61] Darko Zikic, Ben Glocker, Ender Konukoglu, Antonio Criminisi, C. Demiralp, J. Shotton, O. M. Thomas, T. Das, R. Jena, and S. J. Price. Decision forests for tissue-specific segmentation of high-grade gliomas in multi-channel mr. In *Proceedings of International Conference on Medical Image Computing and ComputerAssisted Intervention (MICCAI)*, page 369376, 2012.

[62] Darko Zikic, Yani Ioannou, Antonio Criminisi, and Matthew Brown. Segmentation of brain tumor tissues with convolutional neural networks. In *Proceedings of MICCAI workshop on Multimodal Brain Tumor Segmentation Challenge (BRATS)*, 2014.

Curriculum Vitae

Name:	Zhenyi Wang
Post-Secondary Education and Degrees:	University of Western Ontario London, ON, CA M.Sc. in computer science, 2016 - Present
	Northeastern University Shenyang, Liaoning, China B.E. in Digital Media Technology, 2011 - 2015
Honours and Awards:	WGRS, Western University, 2016 -2017 Outstanding Graduate of Liaoning Province, China, 2015
Related Work Experience:	Teaching Assistant, University of Western Ontario, 2016 -2017 Research Assistant, Computer Vision Group, University of Western Ontario, Sep. 2016 - Present