TDT4501 - Computer Science, Specialization project

Fall 2017

Skull stripping using deep neural networks

Øystein Aas Eide

Supervisor: Frank Lindseth

Co-supervisor: Ingerid Reinertsen

Co-supervisor: Daniel Høyer Iversen

Submission date: June 6, 2018

Norwegian University of Science and Technology

Abstract

Skull stripping of MRI images is an important preliminary part for many medical applications that is dependent on studying the brain. Accurate predictions can contribute to better and more precise analyses for these studies. For accurate predictions manual segmentation is still the most used to get the best results. These manual segmentation can take 6-8 hours to perform on a $1mm^3$ isotropic volume[12]. The automatic segmentation systems that exists today do not take advantage of the methods and technology that have been developed and improved over the past few years. They also struggle with MRI images taken with different modalities and are not good when presented with otherwise altered tissue[20][12].

In this specialization project a 3D convolutionial network based on the article *A 3D convolutional neural network for skull stripping*[20] was built to perform automatic skull stripping. The network was able to achieve a DICE score of 0.948 using 2-fold cross validation on two data sets. The convolutional neural network is able to do segmentations well when it was trained on data from the same source, but it performs significantly worse when the data is from a source it hasn't trained on. An experiment on batch size was also done. This experiment showed that increasing the batch size didn't have a significant effect on how the network performs. However the learning rate wasn't adjusted for these experiments so this experiment did not show a full picture on how the batch size effects learning in a 3D convolutional neural network.

Contents

List of Figures

List of Tables

Abbreviations

CNN	Convolutional neural network
ANN	Artificial neural network
MRI	Magnetic resonance imaging
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent

1 Introduction

Segmentation of medical images using deep learning has exploded in the recent years[15]. The segmentation tasks are usually done to isolate or find particular regions of interest. Manually segmenting these regions are often very time consuming and different standards on what to include and not to include can cause further problems. Since the possibility of using deep learning for segmentation tasks arised the automatic solutions have often become both easier and more reliable. Deep learning requires no to little tuning when it has been trained properly for the task. It also usually provides very good solutions to a wide area of problems.

Skull stripping is a medical imaging task concerned with isolating the brain from other parts of a MRI scan. Isolating the brain to analyze it is so important that a myriad of methods have been developed over the years[12]. Many of these methods perform poorly when confronted with different MRI modalities and brains that are deformed in some way (tumors, Alzheimer's etc.)[20][12]. There is therefor a need for a more reliable automatic segmentation method.

The paper Deep MRI brain extraction [20] presents a solution using a Convolutional Neural Network. These types of neural networks have performed very well on tasks such as image segmentation, image predictions and tumor segmentation. Therefore they should be well suited for skull stripping. The method scores high when evaluated against three different public data sets and it beats the existing methods on most of the tests that were conducted.

1.1 Goals and research questions

The goal of this specialization project is to perform automatic segmentation of the brain on MRI images. The solution for this should be based on the solution from [20], but it will be implemented in a different framework. One experiment will by tried out to improve upon the method by using a larger batch size. The solution will be tested on two publicly available data set: OASIS[13] and LBPA40[34]. There will also be some testing on data from St. Olavs hospital. Since there are no labels available for this data, no thorough results and discussion will be had regarding this. The goal is also to see how the method performs when confronted with data it hasn't seen yet. This can give a clue on how it will perform on the St. Olavs data when labels for it are available.

Therefore the research questions are as follows:

1. To what degree does the batch size effect the results of skull stripping when using deep learning?
2. How good are brain extraction predictions when you train a model on data from one source and test the model on data from another source?

1.2 Structure

This specialization project thesis is divided into 7 chapters. The first chapter gives an introduction to this project and it also presents the goals and research questions. The second chapter presents an introduction to the theory. This includes an introduction on MRI and skull stripping, a description of neural networks, a look at what previous work has been done using deep learning for segmentation tasks and a presentation of the article that this specialization project is based on. Chapter 3 describes the methods used to conduct the experiments, this includes the set up, description of data sets, how the experiments where set up with regards to hyperparameters and which methods where used to evaluate the experiments. Chapter 4 shows the results and in chapter 5 the research questions are discussed in light of the results. Chapter 6 is the conclusion and chapter 7 proposes future work that can be done.

2 Background

In this chapter some basic theory of the subjects covered in this specialization project are described.

2.1 MRI

MRI is used to generate images of organs in the body. It uses a strong magnetic field to align hydrogen atoms inside the body[9]. Next it uses radio frequency energy to excite the atom. When this stops, the hydrogen atoms return to their resting alignment and in doing so they emit radio frequency energy. This energy is then read by antennas in the MRI machine. A grey scale image is produced by the machine where different shades correspond to different frequencies read by the antennas. The different shades will highlight different tissue types in the scan.

By varying the time between each pulse of the radio frequency energy and varying the time for when to receive the echo from the atoms, different tissue types are highlighted. The time between pulses is called Repetition time (TR) and the time between sending the pulse to reading the echo is called the Time to Echo (TE). Three different modalities are commonly used for creating MRI scans:

- T1- short TR and TE
- T2- long TR and TE
- FLAIR- very long TR and TE

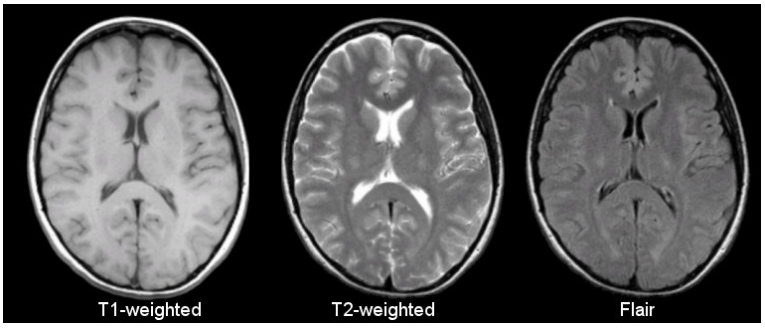

Figure 2.1: Differences in MRI modalities

Source: [9]

The MRI machine can produce 3D images by scanning slices of a given size and then combining them. There are usually some noise in the scans. One of the most common one is variation in the magnetic field. This can be corrected using something called bias field correction[7]

2.2 Skull stripping

Skull stripping or brain extraction is important for many medical fields that study the brain. Some of the most important are:

- Surgical planning
- Cortical structure analysis [26]
- Cortical reconstruction [11]
- Thickness estimation [10]
- Image registration [6]
- Tissue segmentation [27]

Many of these are dependent on accurate estimations of the brain mask to be useful. Therefore impartial, automatic and accurate tools for skull stripping are needed. An incorrect brain mask can cause the patient to be removed from further analyses. With the already sparse availability of data in the field this can be very expensive. The studies are also dependent on their reproducibility and when humans still are the most accurate way of getting predictions the reproducibility can be hampered.

Brains can vary significantly between individuals. Tumors and other anomalies can cause such high differences in the images that it has a serious effect on the predictions of some of the tools that exist today [12][8]. Other factors for differences in the images are: age, different procedures in procuring the images, different scanners, noise in the data etc.

Definition of brain mask

There are no standard for what a brain mask is. There is yet to be an agreement across the board on what should be included and what should be excluded. The definition from [12] includes these for the brain mask:

- *All cerebral and cerebellar white matter*
- *All cerebral and cerebellar gray matter*
- *CSF in ventricles (lateral, 3rd and 4th) and the cerebellar cistern*
- *CSF in deep sulci and along the surface of the brain and brain stem*
- *The brainstem (pons, medulla)*

And it excludes these:

- *Skull, skin, muscles, fat, eyes, dura mater, bone and bone marrow*
- *Exterior blood vessels — specifically the carotid arteries, the superior sagittal sinus and the transverse sinus*
- *Exterior nerves — specifically the optic chiasms*

According to [19] there are two major schools of thought when it comes to what to include and what not to include. These are illustrated in 2.2

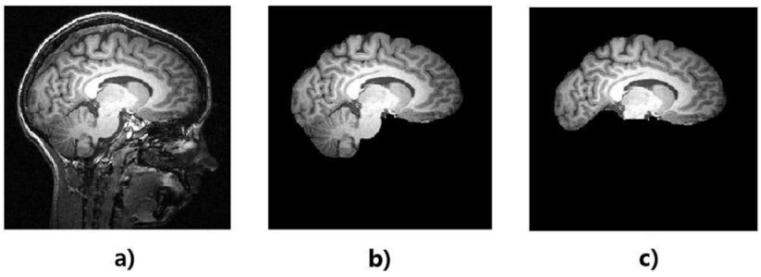

Figure 2.2: a) Full slice of a MRI scan b) Skull stripped brain including the cerebellum and the brainstem c) Skull stripped brain without the cerebellum and the brainstem

Source: [19]

2.2.1 Previous approaches to skull stripping

There have been proposed more than 20 methods for skull stripping[12]. These tools are based on many different techniques and ideas. Most of the methods are designed to be used on T1-weighted MRI scans. This can however cause problems since T1-weighted scans can vary very much between different scanners [12]. Many of the existing methods

have a trade off between being good at removing non-brain tissue and being good at including brain-tissue [8].

According to [20] some of the most popular methods used for skull stripping are:

- BET: Uses a deformable model that starts at the center of gravity for the brain and then it is grown until it fits the borders of the brain [28]
- Hybrid watershed algorithm: uses a deformable surface model combined with watershed algorithms [14]
- 3dSkullstrip: Similar to BET, but also uses points outside the surface to grow the deformable model.
- Brain surface extractor: Uses a region proposal algorithm that is supposed to work as a edge detection for finding the brain mask [25]
- ROBEX uses random forests and a point distribution model [19]
- BEaST: Uses the sum of squared differences to estimate the brain mask based on a library of priors[12]

BET is the most widely used tool today for quick brain extraction [12], it is easy to use and it produces satisfactory results. However when it's faced with tumors it can fail at producing usable brain masks [36]. In fact Hybrid watershed algorithm, 3dSkullStrip, BSE and BEaST are also reported to perform bad when faced with tumors[36][20][12]. Furthermore all except ROBEX and BEaST require parameter tuning [20]. This can cause problems with reproducibility and it can introduce additional labor for finding the right tuning to produce a satisfactory prediction. All of these, except BET, are designed to specifically tackle T1 weighted MRI scans.

2.3 Deep neural networks

Neural networks have been used to solve many problems the past few years. Neural networks usually demands large computing power to both train and predict. Since computers have become better over the past few decades the use of neural networks has become a realistic approach to solving various tasks instead of using conventional algorithms. Where as algorithms are designed to solve specific tasks given a problem a neural network is used to solve a task by first training the network on a set examples and the corresponding solutions to these examples. The model will then "learn" how to solve the problem and hopefully it will also learn how to solve yet unseen examples. This relieves a lot of work from the the programmer. For a conventional algorithm a programmer usually had to engineer features to modify the input so that the program would work satisfactory. A neural network is said to learn these features by itself during training, thus removing the need for hand crafted feature engineering.

The theory of this section is taken from the book Deep learning [17] and the book Neural Networks and Deep Learning [23]. This section will only give a brief presentation on relevant topics for this project.

2.3.1 Artificial neural network

An ANN consists of an input layer and an output layer, it can also consist of, and is most of the time, hidden layers. Each layer is connected to the previous layer and the next layer where each connection has a weight parameter that can be modified. Each layer consists of neurons that take input from the previous layers and produces output to the next layer. In a fully connected layer a neuron has a connection with every neuron in the previous layer and the next layer. See figure 2.3 for an example.

The input to the ANN can f.ex. be an image where the network is supposed to classify what the image contains. Each pixel is then fed to one neuron. The neurons then produces output to the next layer. The neuron in the next layer gets a weighted input from every neuron in the previous layer. This continues until the input has been fed through each layer in the network. The network then produces a classification of the image.

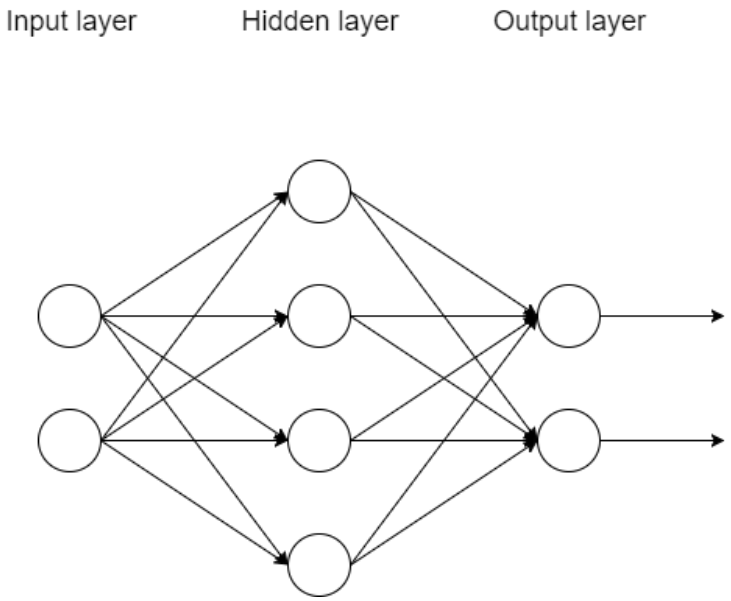

Figure 2.3: A simple neural network

The output of a single neuron is given by this formula [23]:

$$a_j^l = \sigma(\sum_k w_{jk}^l a_k^{l-1} + b_j^l) \tag{2.1}$$

Here w^l_{jk} is the weight from neuron k in layer $l-1$ to neuron j in layer l. a^{l-1}_k is the output from neuron k in layer $l-1$ and b^l_j is the bias for neuron j. σ is the activation function for the neurons. Figure 2.4 shows how the neuron takes input from every neuron in the previous layer and produces an output.

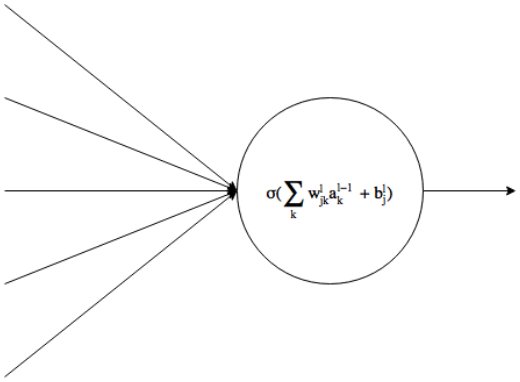

Figure 2.4: Example of a single neuron.

This formula can be rewritten into using vectors and matrices:

$$a^l = \sigma(w^l \vec{a^{l-1}} + \vec{b^l}) \tag{2.2}$$

Here w^l is the weight matrix for layer l. Each row in the matrix corresponds to the weights that are input to neuron j in this layer and each column corresponds to the weights going from neuron k in the previous layer. a^{l-1} is the vector of every output from neurons in layer $l-1$ and b^l is the vector of biases in layer l. This notation is very useful for computers since they are very good at matrix computations.

2.3.2 Activation functions

Activation functions are functions that the neuron uses to compute its output. They take the biases from the current neuron, activations and weights from the previous layer and produces and output. It is represented by the σ in formula 2.1 and 2.2.

Rectified Linear Unit

ReLU is a popular activation function today. The formula is given by:

$$\sigma(z) = max(0, z) \tag{2.3}$$

One drawback about the ReLU function is than when z is negative it will have a gradient of 0, meaning that it will not update the weights. The neuron is then said to be dying. This can be combated with choosing a smaller learning rate so that the update steps aren't large enough to kill the neuron.

Softmax function

The softmax function is given by:

$$\sigma(\vec{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} for j = 1, ..., K \tag{2.4}$$

$\vec{z}$ is the vector of outputs from a given layer. The function outputs a vector of values between $[0, 1]$ that sums up to one. The function is often used in the last layer of neural networks. When that is used the output can be looked upon as probabilities of which class the input belongs to. In other words this means that low output from a neuron in the output layer means that there is a low probability that the input belongs in this class and vice versa.

Absolute value function

This function is very similar to the ReLU function. The function is given by:

$$\sigma(z) = |z| \tag{2.5}$$

This means that it is equal to the ReLU function when the output is positive. However when z is negative it will output the absolute value of z. A positive of this is that the gradient will not be 0 when z is 0 like it would for ReLU.

2.3.3 Cost function

Cost function or loss function is used to give an estimate to the network on how good the predictions in the training phase are[23]. Mean-squared error is one of the simplest loss functions and it is given by this formula:

$$MSE = \frac{1}{n} \sum_{x} (y(x) - a)^2 \tag{2.6}$$

Here MSE is a function with the weights and biases as parameters. $y(x)$ is a vector of the desired outputs given training example x and a is the vector of predicted outputs from the model. This equation is big when the prediction is bad and small when the prediction is good. So object of the ANN is to minimize MSE over all training examples.

Kullback-Leibler Divergence

The formula is given by:

$$D_{KL}(P||Q) = -\sum_i P(i)log(\frac{Q(i)}{P(i)}) \tag{2.7}$$

The formula gives a distance measure on how different probability distribution Q is from probability distribution P. From the formula one can see that if they're equal the formula outputs 1. If P and Q are not equal it will output some value between $[0, 1 >$.

2.3.4 Training

During training the network first feeds the input through the network. With the output it calculates the cost function. Then it tries to minimize the cost function by updating the weights and biases using an algorithm called *Stochastic gradient descent*. Other ways of updating also exist, but this one is the most basic. *SGD* is based on computing how much each parameter contributes to the cost function. It does this by computing the gradient for every parameter. Each parameter is then updated by the negative of the gradient. In other words given parameter θ it computes how much θ contributes to the cost functions. Then it updates θ by the negative of $\frac{\partial C}{\partial \theta}$.

To compute the gradients for all the parameters in the network SGD utilizes an algorithm called *backpropagation*. In short backpropagation works by computing gradients for the output layer and then use those gradients to compute the gradients for the previous layers etc. SGD then has all the gradients and it can update the parameters.

The weights and biases are updated by a small amount that is regulated by a *learning rate* that is a value between $[0, 1]$. Over time the network will hopefully have updated its parameters so that it approximates the problems underlying function. The learning rate regulates how fast the network learns. If it is small the network will learn slowly and if it is large it will learn faster. The downside of having a large learning rate is that it can cause training to diverge. The updates on the weights are then too large causing the predictions of the network to fluctuate wildly from update to update. In the worst case this will make the network never generalize for the training set.

Instead of updating the network after seeing just one example or the whole training set, the model can update the weights after seeing n examples and then updating the weights based on an average of these gradients. This is called *mini-batch gradient descent*. During training this can make the cost-function fluctuate a bit less since it updates the weights after seeing a more complete picture of the problem. And it can also cause the network to learn faster compared to updating after seeing the whole data set since it is computing less gradients per update.

One typically chooses a batch size as a power of 2. They offer a better runtime especially when using GPUs. When a larger batch size is used more memory is needed by the set up to keep them in memory.

Patch wise training

As an alternative to training the network by feeding it whole examples, the network can be fed with only a section of the example. Given an image of size MxM the network can be fed with randomly sampled sections of the image of size NxN where $N < M$. For tasks where the network should predict a class for each pixel or voxel this can be a good approach. Usually the network will then predict some pixels or voxels for the center of the patch. Eg. for a MxM image and a NxN patch the network will predict a OxO pixels where $O < N$.

2.3.5 Adaptive learning rates

With adaptive learning rates the network has one learning rate per parameter. This enables the learning algorithm to tailor the size of the learning rate for each parameter according to how important they are. Different methods for calculating how much each parameters should be updated exist. Adagrad is one of the simplest methods. It stores all the past gradients for the parameters and uses them to scale the learning rate inversely proportional to the square root of the sum of these gradients. AdaDelta is an extension of this. Instead of storing all the past gradients it stores the gradients for a given window size. RMSprop is another method. It adapts the learning rate by decaying the past gradients exponentially. Adam is one of the most recent ones and it is by some considered to be the best one to use today. Adam also takes the second order of derivatives when updating the learning rate. So if the second order is positive it can take a larger step and if it is negative it takes a shorter step.

2.3.6 Convolutional neural network

Convolutional neural networks are a different approach to solving problems compared to traditional neural networks. Whereas the traditional approach uses fully connected layers a CNN uses convolution so that it has less parameters to deal with when training and predicting. It has less parameters because the the neurons share weights and the neurons from layer to layer are not fully connected. This drastically reduces the amount of parameters needed in a CNN. A convolutional network can consists of three parts: convolutional layers, pooling layers and fully connected layers. A neural network that consists of just 1 convolutional layer is still said to be a CNN. Backpropagation with gradient descent is still used to update the parameters.

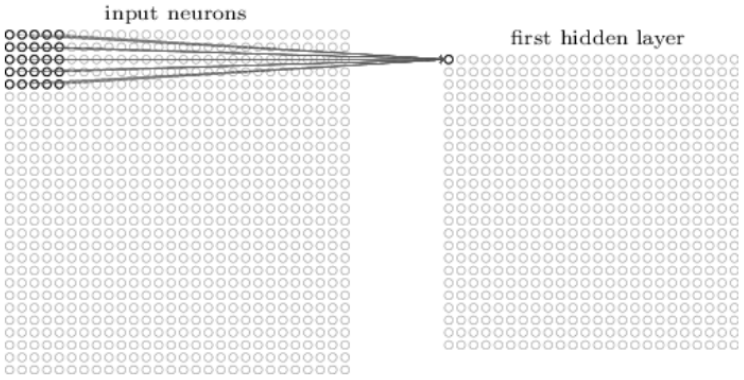

Figure 2.5: CNN example

Source: [23]

2.3.7 Convolutional layers

This formula gives the convolutional operation when the input is 2-dimensional:

$$a_{j,k} = \sigma(b + \sum_{l} \sum_{m} w_{l,m} a_{j+l,k+m}) \tag{2.8}$$

In the equation b is the bias, w is the shared weights and a is the output from the neuron in the previous layer. The equation gives the input to a neuron in the next layer.

This operation is slided across all the input. The output is then said to be a *feature map* and the weights and the bias for this operation is said to constitute a *kernel*. If a kernel is $5x5$ the neuron in the next layer will be calculated using 25 neurons in the previous layer. The 25 neurons is then called the *local receptive field* for the neuron. Figure 2.5 illustrates how the input to one neuron in the next layer is computed. The neurons to the right in the picture constitute one feature map and the connections between the neurons constitutes the kernel.

The slide can be of varying size. The size of the slide is called the *stride length*. If the stride length is 1 the kernel will move over the input by going 1 step at a time. If it is larger it will take larger steps.

A layer can produce many feature maps by using different kernels. When the CNN is trained the feature maps can illustrate "features" in the input that are present. F.ex. if the CNN is used to predict if the input is a face or not, one feature map can detect where eyes appear, while another can detect if there is a nose and so on. In the later layers the features can become more and more abstract. The feature maps output from one layer constitutes the whole input to the next layer.

Since CNNs uses its kernels to compute the next layer the input size doesn't matter. However the output size of the CNN will differ when it has different input sizes. This

can be a problem if the network has a fully connected in it, since fully connected layers can only take one input size.

Up till now only 2D CNNs have been illustrated. It is however trivial to expand the CNN to take 3D input and output. The only difference is that the kernels used are $3x3$ instead of $2x2$ and the feature maps that are output are also 3 dimensional.

2.3.8 Pooling layers

Pooling layers is another method that the CNN uses so that it has less parameters. In the pooling layer one output from a set of neurons of the output from the previous layer is computed. This is done without using parameters. Pooling is also slided across the input akin to the convolution. Different kinds of pooling exists: Max-pooling, average pooling, sum pooling etc. The most often used is max pooling. Max-pooling only takes the largest value in a set of outputs from the previous layer. It is then said that max-pooling only takes the most important features from one layer to the next layer.

2.3.9 Padding

When convolution is used the output is usually smaller than the input. To combat the decreasing size of the output from layers one can apply padding. When padding is used the input is padded with values so that more information is preserved when computing the output. The usual approach is to pad the input with zeros, this is then called zero padding. F.ex. when a $30x30$ image is convolved with $5x5$ kernel and a stride length of 1 the output size is $26x26$. If the image is padded with zeros so that it is $34x34$ then the output becomes $30x30$.

2.4 Generalization and overfitting

A machine learning method is said to generalize well when it makes good predictions on input it hasn't seen before. The opposite of this is when the method is overfitting. Formally this can be expressed as:

- $Error_{train}(H) < Error_{train}(H')$
- $Error_{testing}(H) > Error_{testing}(H')$

H overfits the training data set because there exists an alternative H that has a larger error on the training set, but a lower error on the test set.

Overfitting is a big problem in machine learning. The best way to prevent overfitting is usually to get more training data, but this is many times not possible to do. Regularization techniques are a set of methods used for avoiding overfitting without the need to

expand the training set or changing the network architecture. There exists many types of regularization techniques, and it oftentimes can be difficult to choose the right one for a given problem. Some of them also introduce extra hyperparameters that have to be tuned so that it is adapted to the problem.

2.4.1 Data augmentation

Augmenting the training data is a regularization technique where the training data is altered slightly. With this method the training data can be artificially expanded many fold. The reason why this works is that although a slightly changed input may look the same to a human eye, and therefore we want to classify it the same as the non-augmented data, it can cause the deep learning method to process it very differently.

2.5 Training, testing and validation data

It is conventional to split the data available into a training set and a testing set. The training set is then only used to train the model and the test set is used to evaluate the model. Furthermore the data set can be split into a a validation set. The validation set is usually used as a set to tune the hyperparameters of the machine learning method. When a machine learning methods has a low error rate on the training data, but a high error rate on testing data the model maybe overfitting the training data.

2.6 Cross-validation

Usually data and labels for neural nets are very hard to procure. If a model is to be tested on how well it generalizes by giving it unseen data it can be difficult when not much data is available. For a given task it can then be hard to estimate with statistics which model that will be best suited for real world applications. To rectify this one can use "cross validation". In k-fold-Cross validation the data is split into k equal sized partitions. The model is trained on k-1 partitions and then tested on 1 partition. This is repeated k times so that the network has trained and tested on all the partitions. The accuracy for each of these training examples are then averaged to get an estimate on how well the model generalized for the task.

2.7 Previous work on medical image segmentation using deep learning

The use of deep learning has exploded in the field of medical image analyses. Figure 2.6 shows this in the form of a sudden surge of papers published on the topic. [15]

provides a summary for over 300 contributions in medical image analyses based on deep learning. The paper includes a section about segmentation, but it doesn't mention any approaches to skull stripping. Although the article this specialization project is based upon [20], is listed as one of the articles that have been published in the field of medical image analyses. Searches on Google scholar [3] for the search terms: "brain extraction deep learning" and "skull stripping deep learning" yielded only [20] as a paper that was trying to solve skull stripping using deep learning.

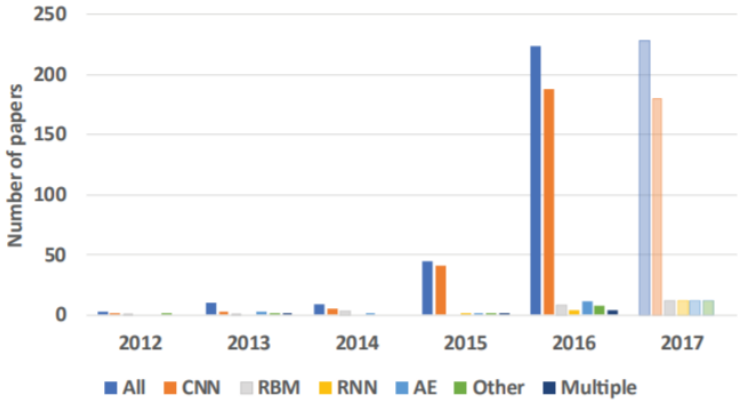

Figure 2.6: Papers published for medical image analyses based on deep learning

Source: [15]

Skull stripping is probably most similar to tumor segmentation since they both are problems that are concerned with doing segmentation of an MRI scan of the brain. In the brain tumor competition BRATS [1] deep neural networks have won the last few years. [15] lists many examples of deep neural networks used for tumor segmentation.

One example is *Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks* [16] which uses an architecture called U-Net [24] to perform segmentation of tumors. U-net is a CNN that first perform regular convolutions on the input followed by upsampling that increases the size of the image. There are equal parts convolutions and upsampling and the layers are also connected with something called skip connections. With this the architecture could process the whole image in one forward pass of the network. One of the advantages of this is that the architecture can exploit all of the data in the image. This is in contrast to patch wise training where the model can only process parts of data in an image. Another example is *DeepMedic for Brain Tumor Segmentation* [21] a 11-layers 3D CNN. The CNN takes two inputs, one high resolution and low resolution, and computes them in parallel. This allows the CNN to have a large receptive field and it allowed to the network to be deeper. The network is based on an earlier version[22], but it has been extended with residual connections, further increasing the conceivable depth of the network.

2.8 Article: Deep MRI brain extraction: A 3D convolutional neural network for skull stripping

The basis for the article is to find a better method for skull stripping. The goal is to produce a skull stripping method that is near parameterless and that works on all modalities of MRI. The article propose that using deep learning might be very suitable solution for this task, since deep learning has been used successfully for many similar tasks in the past years. Specifically they chose to use a CNN, because it has already proven to be well suited for image recognition tasks.

An experiment on extracting brain masks from scans with brain tumors where also conducted, but that won't be covered here.

2.8.1 Architecture

The model consists of 8 convolutional layers where the first 7 layers uses the absolute value function and the last layer is a soft-max output-layer. The full architecture is shown in 2.1, but it emits the max-pooling layer which sits between layer 1 and 2. The max-pooling layer has a stride of 2 and uses a 2^3 window. The model uses Kullback-Leibler divergence as the cost function and the network takes patches of the data for training and prediction.

Layer	1	2	3	4	5	6	7	8
Kernel size	4	5	5	5	5	5	5	1
n feature maps	16	24	28	34	42	50	50	2
Input size n^3	65	31	27	23	19	15	11	7
Layer output size n^3	31	27	23	19	15	11	7	7

Table 2.1: CNN architecture for [20]

The model predicts 1 voxel for a 53^3 voxel input. I.e. the receptive field of the CNN is 53^3. This means that if the input size of the model is n the predicted voxels become:

$$\frac{n-53}{2}-1 \tag{2.9}$$

The network is trained using SGD and a learning rate of 0.000001. The learning is halved if the loss hasn't decreased in 5000 update steps and the learning rate hasn't been updated for 4000 steps. The training is ended after the 10th reduction. A mini batch size of 4 is used.

The program was implemented in python 2.7 using Theano[5]. However 3D convolution and max-fragment prediction (see section 2.8.3) had customized implementations.

2.8.2 Training

For training a random patch from the data is selected. The article says that a input shape of (65, 65, 65) is used, but according to the code the input is really dependent on the size of the MRI scan. The label patch is selected so that it is the center of the selected input patch. The size of the label is given by formula 2.9. For training SGD is used, but looking at the code they have experimented with many different approaches including AdaDelta and Rprop.

Pre-processing

In the article it is mentioned that the data is rescaled to $[0.0, 1.0]$. However the code rescales the data according to this formula for input I:

$$I = \frac{I - mean(I)}{4std(I)} \tag{2.10}$$

Here $mean$ is the mean of input I and std is the standard deviation. This prevents outliers from influencing the prediction to much. The method also adds a random value in range $[-0,05, 0.05]$ and multiply the data with a value in range $[0.85, 1.3]$. This makes the model not focus on the values themselves, but how the values relative to each other should be labeled. With this the model is more robust against different procedures for taking MRI-scans.

2.8.3 Prediction

For predictions patches are fed from the input into the CNN. The CNN then outputs a prediction for the central voxels of the input patch. The predictions for the patches are then used to build the full brain mask. Since only the central voxels for each input are predicted padding is applied to the data so that the voxels at the edges of the image can also be predicted. The output is a 3D tensor where each value is a probability of there being a brain-tissue voxel there. The brain mask is computed by only selecting probabilities over a threshold. 0.5 is used in the code.

For post-processing a function is used for stripping predicted brain-tissue that is not part of the largest predicted brain-tissue component. This is just a safety measure since the program usually outputs a single mask for its predictions.

For prediction the method also uses something called max-fragment-pooling[18]. This was used so that were generated more predicted voxels per forward pass. This was not used during training since it added a significant amount of computational load.

2.8.4 Results and discussion

The article compares its solution against 6 popular and freely available skull stripping tools: Brain extraction tool (BET), Hybrid Watershed Algorithm, 3dSkullStrip, Brain surface extractor, ROBEX and BEAST. 3 data sets where used to train and test the model: OASIS [13], LBPA40 [34] and IBSR [4]. 2 experiments where conducted. One used 2-fold cross validation and on the other experiment models where trained on one data set while predicting on the two other. See table 2.2 for their results on 2-fold cross validation.

Dice coefficient	Sensitivity	Specificity
95.77($\pm$0.01))	94.25($\pm$0.03)	99.36($\pm$0.003)

Table 2.2: Results for [20] using 2-fold cross validation

The article says that the model can be used on all modalities of MRI and that it even can be used on other scanning methods such as CT. As for the results in the first experiment the highest specificity for all data sets and the highest DICE score for the IBSR and the LBPA40 data set was achieved. The second highest DICE score was achieved for the OASIS data set.

In the article it is suggested that the model can be used in other institutions, but the network should be trained on similar examples that the institutions uses to give good results. Even just one example can improve the accuracy significantly.

3 Method

This section describes the implementation of the CNN used to predict brain masks, the data sets, the different experiments and the set up used to conduct the experiments.

3.1 The CNN model

The CNN model is based off the model presented in [20], it is described in section 2.8. The model has the same number of layers and each layer has the same number of feature maps and the same kernel sizes. The same pre-processing methods are also used. Instead of using SGD this method uses the Adam optimizer. This was done since it is reported that it generally gives better performance. This model also uses ReLU instead of the absolute value function. [20] says that this should not have a impact on how the network performs. For predictions max-fragment-pooling wasn't implemented. 3.1 shows the architecture.

Listing 3.1: CNN architecture

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 59, 59, 59, 1)     0
_________________________________________________________________
conv3d_1 (Conv3D)            (None, 56, 56, 56, 16)    1040
_________________________________________________________________
max_pooling3d_1 (MaxPooling3 (None, 28, 28, 28, 16)    0
_________________________________________________________________
conv3d_2 (Conv3D)            (None, 24, 24, 24, 24)    48024
_________________________________________________________________
conv3d_3 (Conv3D)            (None, 20, 20, 20, 28)    84028
_________________________________________________________________
conv3d_4 (Conv3D)            (None, 16, 16, 16, 34)    119034
_________________________________________________________________
conv3d_5 (Conv3D)            (None, 12, 12, 12, 42)    178542
_________________________________________________________________
conv3d_6 (Conv3D)            (None, 8, 8, 8, 50)       262550
_________________________________________________________________
conv3d_7 (Conv3D)            (None, 4, 4, 4, 50)       312550
_________________________________________________________________
conv3d_8 (Conv3D)            (None, 4, 4, 4, 2)        102
_________________________________________________________________
activation_1 (Activation)    (None, 4, 4, 4, 2)        0
=================================================================
Total params: 1,005,870
Trainable params: 1,005,870
Non-trainable params: 0
_________________________________________________________________
```

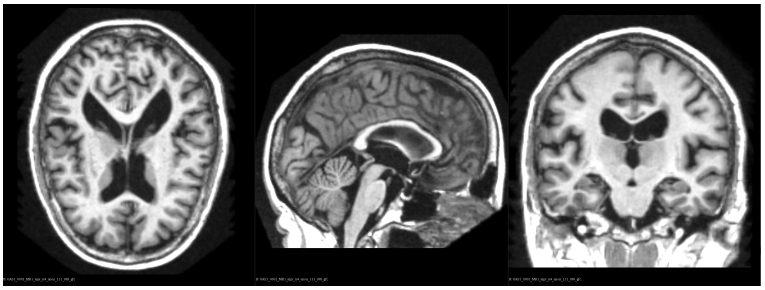

Figure 3.1: OAS1_0001_MR1_mpr_n4_anon_111_t88_masked_gfc_fseg from OASIS data set

3.2 Data sets

3.2.1 OASIS

The first two discs of the OASIS data set were used. Only the first two discs are used because then the result can be better compared with the results in [20]. The discs provide a total of 77 scans. The OASIS scans that were used to train were not raw, they had undergone bias field correction[13]. The target data had skull stripped scans where each brain tissue voxel had a value between 1 and 5. The segmentation was done mostly automatically by programs and then verified by experts. This means that the segmentation is not perfect and may cause the model to have errors in its predictions for other data sets. The scans are from people of ages 18-96 and it includes people that have Alzheimer's disease in the early stages. The scans are T1-weighted and includes both the cerebellum and the brainstem for the brain mask. Table 3.1 gives an overview of which data that was used from the disc for these experiments.

Used for	Name	Dimensions	Vox. size
Data	OAS1_xxxx_MRy_mpr_ni_anon_111_t88_gfc	176x208x176	1x1x1
Label	OAS1_xxxx_MRy_mpr_ni_anon_111_t88_gfc_fseg	176x208x176	1x1x1

Table 3.1: Data used from the OASIS data set

3.2.2 LPBA40

The LONI Probabilistic Brain Atlas [34] contains MRI scans of 40 subjects where the scans have been skull-stripped. This was performed manually. The data used in this specialization project was T1 weighted and it was not bias field corrected. The included brain masks includes both the cerebellum and the brainstem. Table 3.2 gives an overview of which data was used for these experiments.

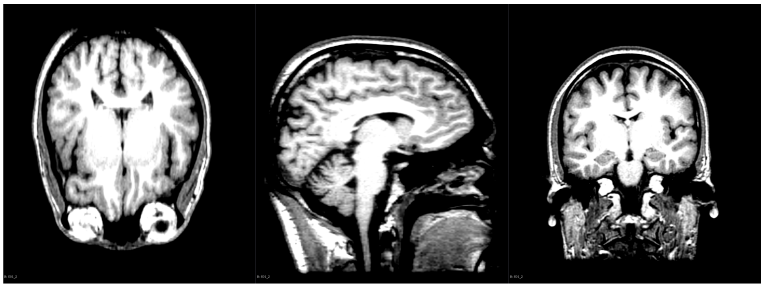

Figure 3.2: S01.native.mri from LBPA40 data set

Used for	Name	Dimensions	Vox. size
Data	S__.native.mri.img.gz	256x124x256	0.859375x1.5x0.859375
Label	S__.native.brain.mask.img.gz	256x124x256	0.859375x1.5x0.859375

Table 3.2: Data used from the LBPA40 data set

The dimensions on the LBPA40 data set had some variations. Some data had dimensions: $256x120x256$.

3.3 Experiments

First set of experiments were done to see how varying the mini batch size would effect the training and accuracy of the network. The second set of experiments is conducted to see how training and predicting on different data sets impact the network.

In the experiments where 2-fold cross validation was used the fold that the model wasn't training on was used to compute validation data and also compute the scores that are described in section 3.4.

3.3.1 Experiment with batch-size

These batch sizes where chosen to see how the network trains and predicts when it has a smaller batch size and when it has larger batch size. It is presumed that there is no need to have a batch size larger than 16 to see a trend for this. Also training on larger batch size takes much longer since you then have to compute gradients for more examples. Table 3.3 shows which batch sizes where used to train the different models.

For this experiment only the OASIS data set was used to train and test on. This was done to simplify the training and adding another data set would most likely do nothing for exposing the trend in training and predicting that is dependent on the batch size.

To test and evaluate the different models 2-fold cross validation was used.

Run	mini-batch size	Loss function
1	4	Kullback-Leibler divergence
2	2	Kullback-Leibler divergence
3	8	Kullback-Leibler divergence
4	16	Kullback-Leibler divergence

Table 3.3: Experiments with batch size

3.3.2 Experiments with different data sets

Experiment 1

This experiment is designed to test how well the model predicts when it is trained and tested on the same data set. Specifically the OASIS data set was used for this. No validation data was used.

Experiment 2

To test how well the method performs when it is used on two data sets the method is trained and tested on both data sets using 2-fold cross validation. Validation accuracy was also logged.

Experiment 3

The method was trained on one data set and tested on the other. This experiment is dependent on how similar the data sets are and the scores will probably reflect this. The data sets have different resolutions and [20] says that this can have an impact on how well the network predicts.

3.3.3 Experiments on St. Olavs data

There are no labels for the St. Olavs data. So it is not possible to produce an objective way of evaluating the performance of the model on the data. Therefore this experiment will only show images of the original scan with the prediction laid over it. The prediction using a model from [20] is also included for comparison. This is possible because in the publicly available code there is also a pre-trained model [31]

3.4 Evaluation metrics

The models predictions were evaluated using three metrics:

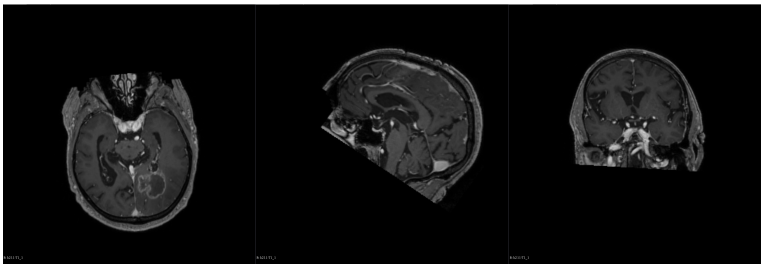

Figure 3.3: k211-T1 from St. Olavs data set

$$specificity = \frac{TN}{TN + FP} \tag{3.1}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{3.2}$$

$$DiceScore = \frac{2TP}{2TP + FP + FN} \tag{3.3}$$

The symbols stand for:

- TP (True positive) correctly predicted brain tissue voxels.
- FP (False positive) falsely predicted brain tissue voxels.
- TN (True negative) correctly predicted non-brain tissue voxels.
- FN (False negative) falsely predicted non-brain tissue voxels.

Specificity gives a score based on how well the model predicts negative voxels for the data. Sensitivity tells how good the model is at predicting positive voxels. The dice score or dice similarity coefficient penalizes the score for having many false positives and false negatives. This score therefore tells a better story on how the model performs compared to just the sensitivity or specificity scores.

3.5 Set up

The experiments where performed on a NVIDIA 1070 GPU and an i7-6700K intel CPU clocked at 4.00GHz. The program was written in python 3.6 using Keras [33] with a Tensorflow-GPU [35] back end. The framework was chosen for its simplicity as well as it was a wish from the people working at St. Olavs for SINTEF. In Keras the model was trained using $fit\_generator$ [32], this enabled using the GPU for training and the CPU for image pre-processing, potentially speeding up the program. For enabling the GPU to do general computing CUDA v8.0 and cuDNN[2] v5.1 was used. Windows 10 was

used as the operating system. The optimal would have been to use Linux since CUDA and cuDNN is easier to set up there.

All the graphs were computed using matplotlib [30] in python. MRI scans were illustrated using 3D slicer [29].

4 Results

This chapter describes the results for the experiments. It is divided in three section: the first one showing how batch-size affects the prediction, the second shows how training on different data sets affects the predictions and the third one which shows images of some predictions on data from St.Olavs.

4.1 Experiment with hyper parameters

The runs for this experiment is presented with:

1. A graph showing how the validation accuracy evolves over the update steps. The y-axis (accuracies) values are are averaged over 500 update steps. This is because the accuracies for updates are volatile since they are from random patches from the input data. The graph will still show a trend on how the model has learned the data. The accuracies are also averaged for the two folds in the 2-fold cross validation.
2. The average DICE score, sensitivity, specificity and time to train. The standard deviation is presented in parenthesis.

Batch size	Dice coefficient	Sensitivity	Specificity	Time to train
2	0.93561(±0.0147)	0.97505(±0.0119)	0.96168(±0.0124)	1762s
4	0.94206(±0.0150)	0.97314(±0.0133)	0.96740(±0.0125)	3033s
8	0.94275(±0.0149)	0.97371(±0.0126)	0.96771(±0.0126)	4740s
16	0.94370(±0.0145)	0.97321(±0.0086)	0.96863(±0.0123)	8858s

Table 4.1: Results for experiments with batch size

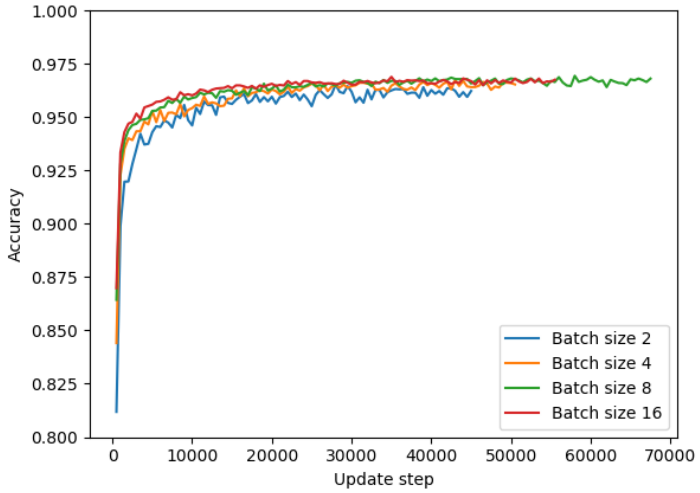

Figure 4.1: Accuracies on validation data for all different batch sizes

4.2 Experiments with different data sets

The tables are presented the same as the first experiment. The y-axis for the graphs are also averaged the same as for the first experiment.

4.2.1 Experiment 1

Dice coefficient	Sensitivity	Specificity
0.94648(±0.0148)	0.97527(±0.0113)	0.96995(±0.0122)

Table 4.2: Results for model trained and tested on OASIS data

4.2.2 Experiment 2

Dice coefficient	Sensitivity	Specificity
0.95003(±0.0128)	0.96790(±0.0139)	0.97552(±0.0115)

Table 4.3: Results for OASIS data with 2-fold cross validation

Dice coefficient	Sensitivity	Specificity
0.94703(±0.0190)	0.96174(±0.0336)	0.98779(±0.0066)

Table 4.4: Results for LBPA40 data with 2-fold cross validation

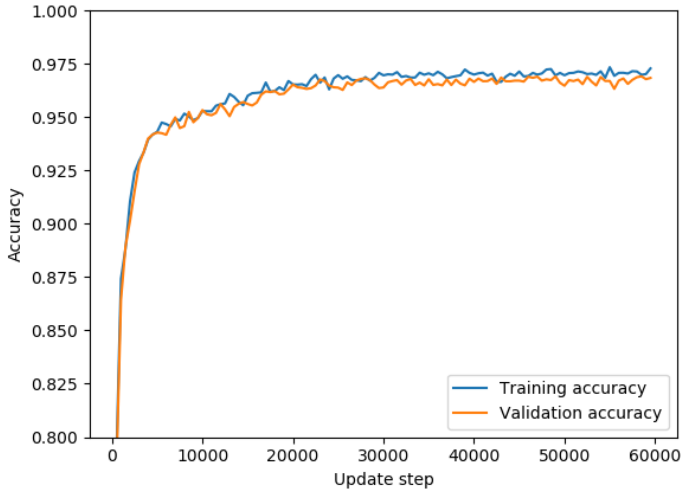

Figure 4.2: Validation and training accuracy for 2-fold cross validation on both data sets

4.2.3 Experiment 3

Two brain mask predictions are also illustrated for this experiment. This is to give an idea on how the models trained on one data sets predicts brain masks for the other data set.

Dice coefficient	Sensitivity	Specificity
0.77691(±0.03881)	0.99524(±0.00327)	0.89759(±0.02171)

Table 4.5: Results model trained on OASIS and tested on LBPA40

Dice coefficient	Sensitivity	Specificity
0.74254(±0.1016)	0.60661(±0.1250)	0.99650(±0.00359)

Table 4.6: Results model trained on LBPA40 and tested on OASIS

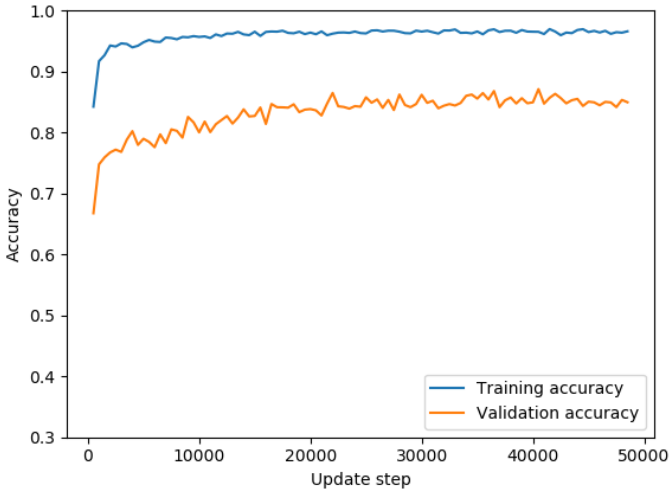

Figure 4.3: Training accuracies and validation accuracies when training on OASIS and validating against LBPA40

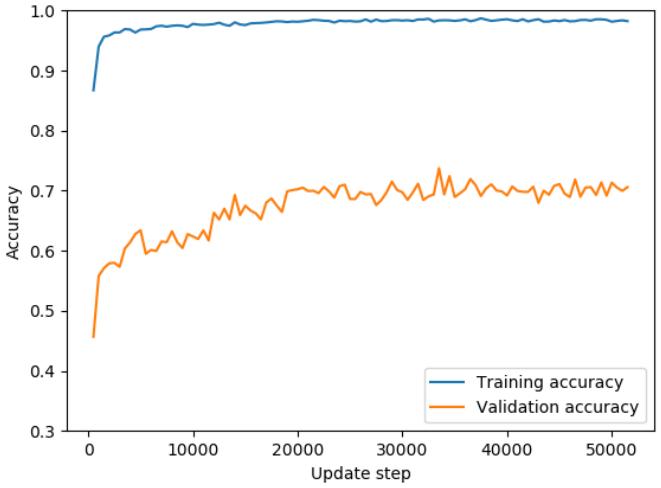

Figure 4.4: Training accuracies and validation accuracies when training on LBPA40 and validating against OASIS

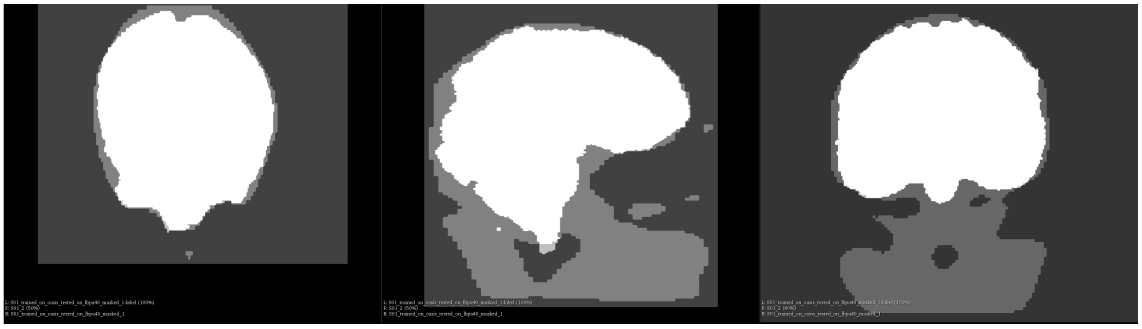

Figure 4.5: S01.native.mri brain mask predicted by model that trained on OASIS. White is the ground truth and grey is the predicted brain mask.

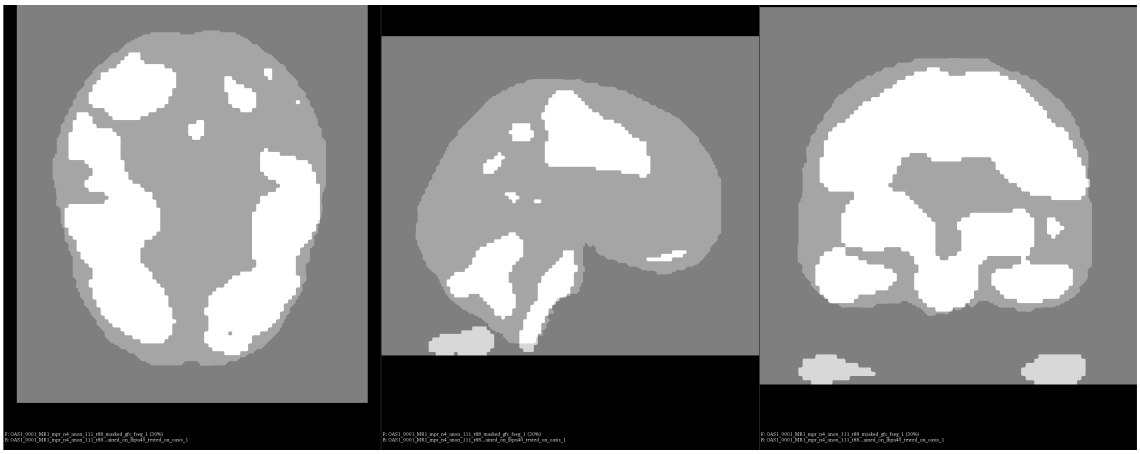

Figure 4.6: OAS1_0001_MR1_mpr_n4_anon_111_t88_gfc brain mask predicted by model that trained on LBPA40. White is the predicted brain mask and grey is the ground truth.

4.3 Experiment on St. Olavs data

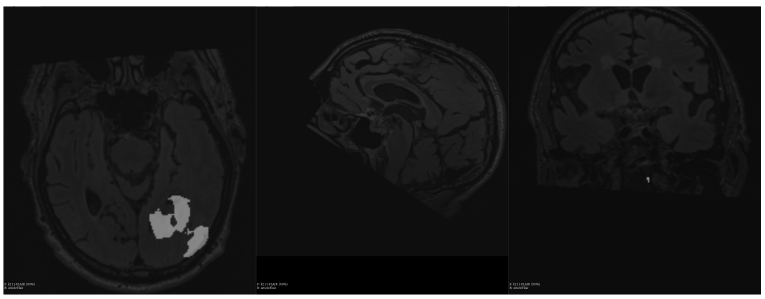

Figure 4.7: k211-FLAIR prediction from [31][20]

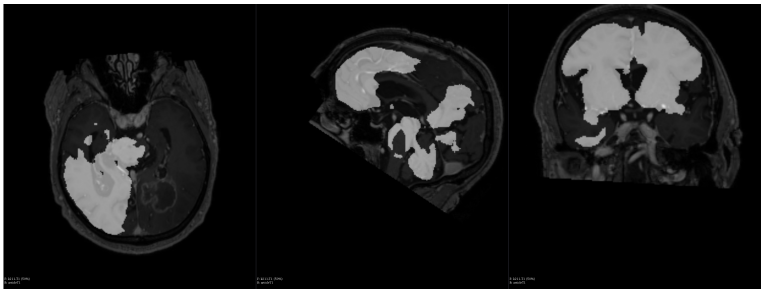

Figure 4.8: k211-T1 prediction from [31][20]

As can be seen in figure 4.7 not many voxels where predicted to be brain tissue using the trained network from [20] on the FLAIR MRI scan. Only 1.3% of voxels where predicted to be brain-tissue. On the T1 weighted image 4.8 much many more positive brain-tissue voxels are present. Here 6.4% of voxels where predicted to be brain-tissue.

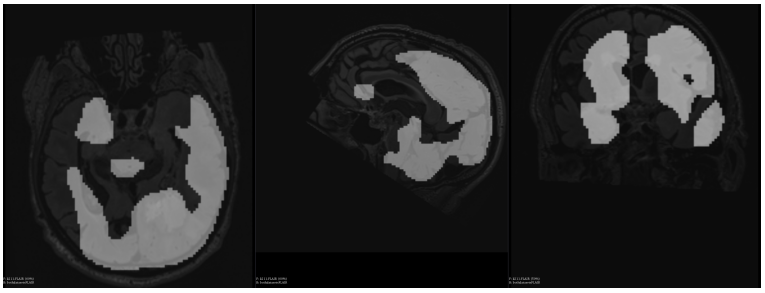

Figure 4.9: k211-FLAIR prediction from model trained on data from OASIS and LBPA40

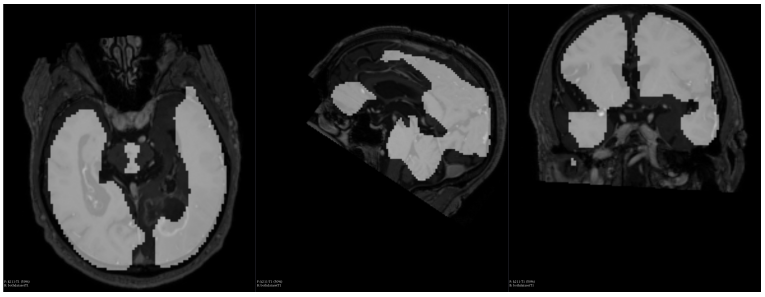

Figure 4.10: k211-T1 prediction from model trained on data from OASIS and LBPA40

The model that is trained on both data sets predicts many more positive voxels. It can also be seen here that the model predicts slightly more positive voxels for the T1 (9.9%) image compared to the FLAIR image (8.0%).

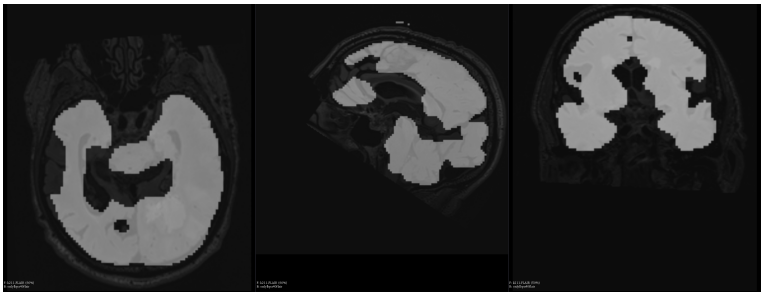

Figure 4.11: k211-FLAIR prediction with model trained on data from LBPA40

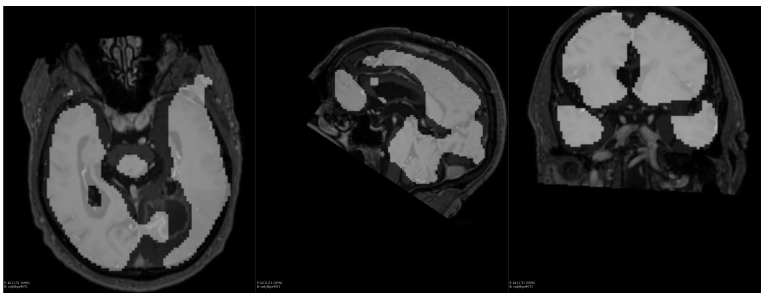

Figure 4.12: k211-T1 prediction with model trained on data from LBPA40

For figure 4.11 and 4.12 the model predicts slightly more positive brain-tissue voxels for the FLAIR (10.6%) than for the T1 (9.9%)

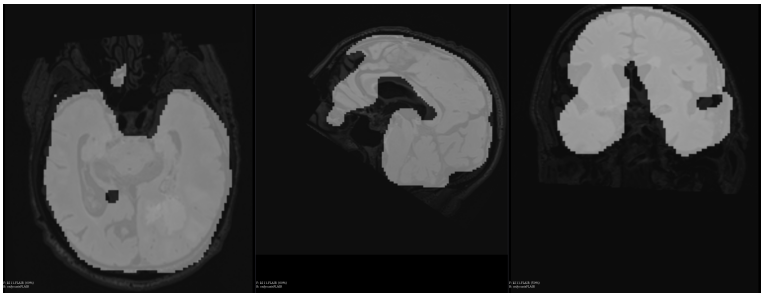

Figure 4.13: k211-FLAIR prediction with model trained on data from OASIS

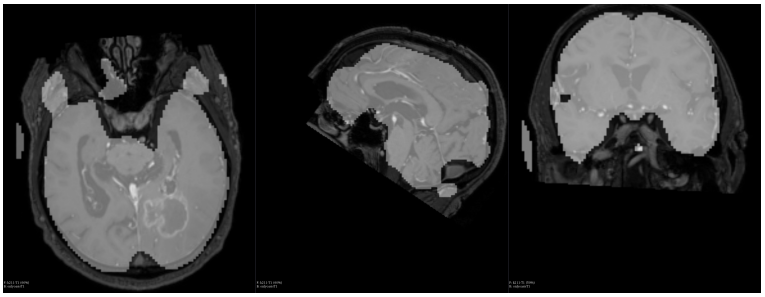

Figure 4.14: k211-FLAIR prediction with model trained on data from OASIS

For these predictions the brain mask seem to follow the edges on some places correctly. The FLAIR has 14.4% positive voxels and the T1 image has 14.1% positive voxels.

5 Discussion

This chapter discusses the results considering the goal and research questions.

5.1 Experiments with hyper parameters

The experiments show a trend that a larger batch size can be slightly better. This is shown in both the DICE score and the accuracy plots. The improvement was however not very substantial and it added a large penalty in the form of more training time. All these results are consistent with the theory behind. With a larger batch size more forward passes are needed and more gradients are computed per update. Overall choosing a mini batch size over 4 doesn't seem to have a huge impact on how the model performs.

With a larger batch size a larger learning rate can be used. This is because when the gradients are averaged the update better approximates the underlying function, and therefor a larger step size for updating the parameters can be used. The learning rate for this method wasn't changed. Since this method was implemented using Adam which has a learning rate per parameter, the effect of tuning the learning rate maybe wouldn't have made a big difference.

From figure 4.1 it can be seen that when the batch size is larger the accuracy of the network is less volatile across update steps. This can be explained by the same reasoning as above. Interestingly when the batch size was 8 the model took longer to finish compared to the other ones. Not much should be read into this however because the network stops training by monitoring if the cost function hasn't been lowered for a certain amount of update steps. So it could be random that for the batch size 8 the cost function reached a low point more at the right time or it reached a low point more frequently. To give any conclusion on this more experiments have to be done.

For all the experiments it seems by looking at figure 4.1 that after update step 20000 the network is overfitting the data. However the evaluation scores in table 3.3 scores showed that the model didn't overfit much, since they were obtained using cross validation. One reason for why it is difficult for the model to overfit is because each each training example is processed by adding and multiplying it with random values. The model is effectively trained on a close to infinite number of different examples. It is then very difficult for the model to learn the noise of the training data. The high score then indicates that this method did well for hindering overfitting. The code in [31] indicate that they had done a lot of experiments with regularization that ultimately didn't end up in the final

model. These results also indicate that there is no pressing need to add regularization techniques.

Stopping the training after fewer update steps could maybe have been possible, especially for the larger batch sizes. From figure 4.1 the accuracies for the validation set is more stable and it is likely that the model would have performed well when stopping earlier. If the model were to stop training earlier the training time would also go down.

The answer to ***research question 1*** is then that a batch size over 4 seems to give little gain, but a much longer training time. The accuracy accross update steps also got a little more stable and therefor the model could maybe have stopped training earlier.

5.2 Experiments with different data sets

Table 4.3 and figure 4.4 shows that the model can generalize well when trained on both data sets. It shows a high score for both the OASIS data set and the LBPA40 data set. The DICE score are a bit higher for the OASIS data set compared to those in table 4.2 and those results are for a model trained and tested on OASIS data. It seems that when data is introduced from another source the DICE got higher, but it got a lower sensitivity score. So the model trained on both data sets has fewer predicted brain-tissue voxels, but it is better at predicting voxels that are not brain-tissue. The LBPA40 data set might have caused the network to be more discriminatory on what to predict as brain voxels. This in turn lead to it having a higher DICE score.

Table 4.6 and table 4.5 shows that the model performs poor when it is predicting data from sources it hasn't seen yet. The sensitivity and the specificity scores show that it's predicts far to many negative voxels in it's predictions. This result is not consistent with [20]. One explanation for this is that that the model had one extra data set to train on. This may have caused the network to be less specialized for the particular the data set.

Interestingly the results for sensitivity and specificity in table 4.6 somewhat mirror the results in table 4.5. One has high sensitivity and the other has low sensitivity and the reverse for specificity. This indicates that the model trained on the LBPA40 predicts far fewer positive voxels while the model trained on the OASIS data predicts to many positives. This can be seen in figures 4.5 and figure 4.6. One reason for this can be what is included and what is not included in the MRI scan. In figure 3.1 and figure 3.2 it can be seen that the LBPA40 scans include everything almost to the neck while the OASIS scans stops over the mouth. In figure 4.5 it can seen that the model predicts many positive in this region that it hasn't seen yet.

The model trained on LBPA40 predicts very few positive brain-tissue voxels. This can be the results of the MRI scans from the LBPA40 data set has higher intensities compared to the OASIS data set. This is also an explanation for why the model trained on OASIS data predicted too many voxels for the LBPA40 data. Changing the threshold for what output probability should constitute a brain voxel might make the predictions

better. The threshold should then be higher for the model trained on OASIS and a lower threshold for the model trained on LBPA40. For data sets where no training examples are available this might be a parameter to tune for giving better predictions.

The results are comparable to the results in [20], but slightly worse. One reason why its worse could be that that method had one more source of data to train on. As table 4.3 and table 4.2 suggests, training on data from other sources made the model get better at predicting on the initial set as well. Other reasons is that the other method also use techniques like max-fragment-pooling and their hyperparameters are probably tuned exactly to work with their optimizer where as this solution has a different activation function and optimizer, but the same learning rate and batch size.

Answer to ***research question 2*** is then that it seems to be very important to train the network on data from the source that it is going to predict on. The standards for taking MRI scans can vary so widely that it is perhaps crucial to train the network on these data. Having data from many sources when training the network also seems to give a better performance on some metrics for the network. It is important to remember that this was only the case for these two particular data sets and when using other data sets this may, and perhaps most likely, isn't going to be the case.

5.3 Experiments on St. Olavs data

There are definite differences in the predictions from the different models on the St. Olavs data. It seems that the model that was trained on only the OASIS data set has the best predictions overall. For the FLAIR image 4.13 it seems that it has many holes in the center of the image and it seems like it has predicted a lot of false negative voxels for the image. The T1 prediction (4.14) looks a bit better, but this seems to also have some false predicted brain-tissue voxels. The predictions from the model trained on the OASIS data set is probably a bit better because the St. Olavs scans and the OASIS scans are more similar to each other than the St. Olavs scans and the scans from LBPA40.

Generally on all of the predictions it seems like the model predicted a lot of false negatives. This is also backed up by the percentages of predicted positive voxels. The percentages however shouldn't be taken as hard evidence since there are no ground truths to calculate what the percentage should really be.

Results from experiment 3 and 4 with both data sets indicate that if the model gets to train on the St. Olavs data it will produce better predictions. Maybe if the network was also trained on other data sets it would generalize better on the St. Olavs data.

6 Conclusion

This specialization project has implemented a method for skull stripping using a CNN based on the article [20]. The method has been evaluated against two publicly available data sets and it has performed on par with the method from the article, but the scores are still a bit lower. The first research question asked how batch size effected the network. Experiments where conducted to see if the method could be improved upon by using a different batch size and how the network would then behave. The scores didn't improve much, but the cost function stabilizes more across update steps. This however came with a huge cost in training time.

The method performed poorly when it was trained on one data set and tested on another. The results for the experiment where the network was trained on both data sets show that the model still can predict well on both data sets as long as it has trained on both. Predictions on data from St. Olavs should therefor be better if the network is trained on them. This answers the second research question.

7 Future work

The architecture for the CNN produced good results. However since there weren't any label data available from St.Olavs it was impossible to produce results from that set. In the future the network should be trained and tested on these data as well.

The results from the experiments with different data sets suggested that intensity values in the scans can have an effect on the output of the network. Some new pre processing steps should perhaps be introduced to scale the data so that these values are less important for the predictions of the network.

In a MRI scan there are usually more brain voxels compared to non-brain voxels. The current sampling only takes a random cube of the scan. This can cause the model to be trained more on brain tissue voxels compared to non-brain tissue voxels. If the sampling of the cubes could be altered so that it with equal probability selects cubes that are brain and cubes that are non-brain, the network might predict better.

Introduction of new processing steps for the data might cause the model to predict better on data it hasn't seen yet. The OASIS data set and the LBPA40 data set currently have different resolution for their scans. The CNN does not use the information about the resolution so it could have a negative impact on its predictions. Therefore the data and labels could be resampled so that they have the same resolution. [21] recommends doing this.

The existing architecture can be experimented more with. [20] used an older generation of GPUs so they had limited options for setting up certain hyper parameters. Mainly how big the input to the CNN should be and how large the batch size should be.

Other architecture should perhaps also be explored. One interesting architecture is the U-Net architecture [24] which showed good results on other medical imaging segmentation tasks. Another one is DeepMedic [21]. Both have implementation are on GitHub so the code can be used to get a better understanding of their respective architecture.

Bibliography

[1] *BRATS*. URL: http://braintumorsegmentation.org/. [Online; accessed 2015-12-09].

[2] *cuDNN*. URL: https://developer.nvidia.com/cudnn. [Online; accessed 2015-12-13].

[3] *Google Scholar*. URL: https://scholar.google.no/. [Online; accessed 2015-12-08].

[4] *IBSR*. URL: https://www.nitrc.org/projects/ibsr. [Online; accessed 2015-12-09].

[5] *Theano*. URL: http://deeplearning.net/software/theano/. [Online; accessed 2015-12-09].

[6] Brian Avants B.T.T. Yeo Bruce Fischl Babak Ardekani James C. Gee J.J. Mann Ramin V. Parsey Arno Klein, Satrajit S. Ghosh. *Evaluation of volume-based and surface-based brain image registration methods*. URL: http://www.sciencedirect.com/science/article/pii/S105381191000114X?via%3Dihub, 2000. [Online; accessed 2015-12-06].

[7] BrainSuite. *Bias field correction*. URL: http://brainsuite.org/processing/surfaceextraction/bfc/, unknown. [Online; accessed 2017-11-13].

[8] Camellia P. Clark Shaunna Morris Amanda Bischoff-Grethe Mark W. Bondi Terry L. Jernigan Bruce Fischl Florent Segonne David W. Shattuck Richard M. Leahy David E. Rex Arthur W. Toga Kelly H. Zou Morphometry BIRN and Gregory G. Brown Christine Fennema-Notestine, I. Burak Ozyurt. *Quantitative Evaluation of Automated Skull-Stripping Methods Applied to Contemporary and Legacy Images: Effects of Diagnosis, Bias Correction, and Slice Location*. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2408865/, 2006. [Online; accessed 2015-12-07].

[9] MD David C Preston. *Magnetic Resonance Imaging (MRI) of the Brain and Spine: Basics*. URL: http://casemed.case.edu/clerkships/neurology/Web%20Neurorad/MRI%20Basics.htm, 2006. [Online; accessed 2017-11-14].

[10] David Avis David MacDonald, Noor Kabani and Alan C. Evans. *Automated 3-D Extraction of Inner and Outer Surfaces of Cerebral Cortex from MRI*. URL: https://

www.sciencedirect.com/science/article/pii/S1053811999905347, 2000. [Online; accessed 2015-12-06].

[11] Daniel Q. Naiman Susan M. Resnick Michael A. Kraut Duygu Tosun, Maryam E. Rettmann and Jerry L. Prince. *Cortical Change in Alzheimer's Disease Detected with a Disease-specific Population-based Brain Atlas.* URL: http://www.sciencedirect.com/science/article/pii/S1053811905006531?via%3Dihub, 2005. [Online; accessed 2015-12-05].

[12] Coupe P. Fonov V. Manjon J.V. Leung K.K. Guizard N. Wassef S.N. Ostergaard L.R. Collins D.L. Alzheimer's disease Neuroimaging I. Eskildsen, S.F. *BEaST.* URL: http://www.sciencedirect.com/science/article/pii/S1053811911010573?via%3Dihub, 2012. [Online; accessed 2015-12-05].

[13] Marcus et al. *CROSS-SECTIONAL DATA ACROSS THE ADULT LIFESPAN .* URL: http://www.oasis-brains.org/pdf/oasis_cross-sectional_facts.pdf, 2007. [Online; accessed 2017-11-13].

[14] E.Busa M.Glessner D.Salat H.K.Hahn B.Fischla F.Ségonnea, A.M.Dale. *A hybrid approach to the skull stripping problem in MRI.* URL: http://www.sciencedirect.com/science/article/pii/S1053811904001880?via%3Dihub, 2004. [Online; accessed 2015-12-08].

[15] Babak Ehteshami Bejnordi Arnaud Arindra Adiyoso Setio Francesco Ciompi Mohsen Ghafoorian Jeroen A.W.M. van der Laak Bram van Ginneken Clara I. Sanchez Geert Litjens, Thijs Kooi. *A Survey on Deep Learning in Medical Image Analysis.* URL: https://arxiv.org/pdf/1702.05747.pdf, 2017. [Online; accessed 2015-12-08].

[16] Fangde Liu Yuanhan Mo Yike Guo Hao Dong, Guang Yang. *Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks.* URL: https://arxiv.org/pdf/1705.03820.pdf, 2017. [Online; accessed 2015-12-09].

[17] Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning.* URL: http://www.deeplearningbook.org/, 2017. [Online; accessed 2017-11-14].

[18] Dan Ciresan Gabriel Fricout Jurgen Schmidhuber Jonathan Masci, Alessandro Giusti. *Max-fragment-pooling.* URL: http://ieeexplore.ieee.org/document/6738559/, 2013. [Online; accessed 2015-12-10].

[19] Paul M. Thompson Zhuowen Tu Juan Eugenio Iglesias, Cheng-Yi Liu. *Robust Brain Extraction Across Datasets and Comparison With Publicly Available Methods.* URL: http://ieeexplore.ieee.org/document/5742706/, 2011. [Online; accessed 2015-12-06].

[20] Hubert A Schwarz D Maier-Hein K Bendszus M Biller A. Kleesiek J, Urban G. *Deep MRI brain extraction: A 3D convolutional neural network for skull stripping.* URL: https://www.ncbi.nlm.nih.gov/pubmed/26808333, 2016. [Online; accessed 2017-11-13].

[21] Sarah Parisot Christian Ledig Aditya Nori Antonio Criminisi Daniel Rueckert Konstantinos Kamnitsas, Enzo Ferrante and Ben Glocker. *DeepMedic for Brain Tumor Segmentation.* URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2017/02/kamnitsas2016brats.pdf, 2017. [Online; accessed 2015-12-09].

[22] Virginia F.J. Newcombe Joanna P. Simpson Andrew D. Kane David K. Menon Daniel Rueckert Ben Glocke Konstantinos Kamnitsas, Christian Ledig. *DeepMedic.* URL: https://ac.els-cdn.com/S1361841516301839/1-s2.0-S1361841516301839-main.pdf?_tid=236b68d8-d8fa-11e7-977e-00000aacb35d&acdnat=1512395631_77842ce7d52fe140382a8575a71e90f1, 2017. [Online; accessed 2015-12-04].

[23] Michael Nielsen. *Neural Networks and Deep Learning.* URL: http://neuralnetworksanddeeplearning.com/, 2017. [Online; accessed 2017-11-14].

[24] Philipp Fischer Olaf Ronneberger and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation.* URL: https://arxiv.org/pdf/1505.04597.pdf, 2015. [Online; accessed 2015-12-04].

[25] J. Malik P. Perona. *Scale-space and edge detection using anisotropic diffusion.* URL: http://ieeexplore.ieee.org/document/56205/, 1990. [Online; accessed 2015-12-08].

[26] Roger P. Woods Chris I. Zoumalan Chris J. Lindshield Rebecca E. Blanton Jacob Moussai Colin J. Holme Jeffrey L. Cummings Arthur W. Toga Paul M. Thompson, Michael S. Mega. *Cortical Change in Alzheimer's Disease Detected with a Disease-specific Population-based Brain Atlas.* URL: https://academic.oup.com/cercor/article/11/1/1/371537, 2009. [Online; accessed 2015-12-05].

[27] M. Arfan Ikram Meike W. Vernooij Monique M.B. Breteler Aad van der Lugt Wiro J. Niessen Renske de Boer, Henri A. Vrooman. *Accuracy and reproducibility study of automatic MRI brain tissue segmentation methods.* URL: http://www.sciencedirect.com/science/article/pii/S105381191000282X?via%3Dihub, 2010. [Online; accessed 2015-12-06].

[28] Stephen M. Smith. *Fast robust automated brain extraction.* URL: http://onlinelibrary.wiley.com/doi/10.1002/hbm.10062/abstract;jsessionid=AF8EC7FB9B51B1472E59A4080278366C.f04t04, 2002. [Online; accessed 2015-12-08].

[29] Unknown. *3D slicer.* URL: https://www.slicer.org/, ... [Online; accessed 2015-12-07].

[30] Unknown. *matplotlib*. URL: https://matplotlib.org/, ... [Online; accessed 2015-12-07].

[31] Unknown. *Deep_MRI_brain_extraction*. URL: https://github.com/GUR9000/Deep_MRI_brain_extraction, 2017. [Online; accessed 2017-12-03].

[32] Unknown. *fitgenerator*. URL: https://keras.io/models/sequential/#fit_generator, 2017. [Online; accessed 2017-12-03].

[33] Unknown. *Keras*. URL: https://keras.io/, 2017. [Online; accessed 2017-12-03].

[34] Unknown. *LBPA40*. URL: http://www.loni.usc.edu/atlases/Atlas_Detail.php?atlas_id=12, 2017. [Online; accessed 2017-12-01].

[35] Unknown. *Tensorflow*. URL: https://www.Deep_MRI_brain_extractionow.org/, 2017. [Online; accessed 2017-12-03].

[36] Leila El-Kara Zhuowen Tu Corey Arnold William Speier, Juan E. Iglesias. *Robust Skull Stripping of Clinical Glioblastoma Multiforme Data*. URL: https://link.springer.com/chapter/10.1007%2F978-3-642-23626-6_81, 2011. [Online; accessed 2015-12-08].