NTNU
Norwegian University of
Science and Technology

Skull stripping MRI images of the brain using deep learning

Øystein Aas Eide

Master of Science in Computer Science
Submission date: June 2018
Supervisor: Frank Lindseth, IDI
Co-supervisor: Ingerid Reinertsen, SINTEF

Norwegian University of Science and Technology
Department of Computer Science

Problem Description

Name	Øystein Aas Eide
Faculty	Faculty of Information Technology and Electrical Engineering
Department	Department of Computer Science
Study program	Computer Science (Master, five years)
Specialization	Artificial intelligence
Start date	31.01.2018
Due date	27.06.2018
Supervisor	Professor Frank Lindseth
Co-supervisor	Ingerid Reinertsen

Title

Skull stripping MRI images of the brain using deep learning

Task Description

By using deep learning this master thesis will try to perform skull stripping on MRI images of the brain. The thesis will try different deep learning architectures and see how well they perform compared to each other. This thesis will also try to provide a working skull stripping solution for the St. Olavs hospital.

Data were provided in part by OASIS: Cross-Sectional: Principal Investigators: D. Marcus, R, Buckner, J, Csernansky J. Morris; P50 AG05681, P01 AG03991, P01 AG026276, R01 AG021910, P20 MH071616, U24 RR021382

Abstract

Skull stripping is the task of finding pixels or voxels that establishes where the brain is in a medical image. It is an important step for many medical applications that analysis the brain. Today manual skull stripping is still the best way to get good and accurate results. Manual skull stripping is a laborious process since it can take between 6 and 8 hours (Eskildsen et al., 2012). There exist many tools for performing skull stripping automatically. Some of these tools are very good, but they can struggle on Magnetic Resonance Imaging (MRI) scans with different modalities (Kleesiek et al., 2016). In this decade deep learning has become very popular in various fields including medical image analysis, which includes tasks such as medical image segmentation. According to Litjens et al. (2017) the amount of published papers on this subject have gone from close to 0 in 2012 to over 200 papers published in 2017. In this thesis three different deep learning architectures will be used to do skull stripping. Also, an experiment on liver segmentation will be done to see if the methods will work on other medical imaging tasks.

The deep learning architectures will be tested on three different data sets containing MRI scans of the brain. One of these data sets contains only patients with tumors. The architectures were trained on different data sets to test how well they will perform when they are faced with MRI scans from different sources. Various experiments will be done to see how training and testing on different compositions of these data sets will make the architectures perform.

Deep learning methods take a long time to train. In this thesis, the deep learning architectures will be tested on different hardware configurations to see how long they take to train on the different set ups. In total four different configurations will be tested with different Central Processing Units (CPUs) and different Graphics Processing Units (GPUs).

Sammendrag

Hjernesegmentering er oppgaven med å finne hvilke piksler eller voxler som tilhører hjernen i et medisinsk bilde. Det er en viktig fase for mange medisinske applikasjoner som er avhengig av å studere hjernen. Manuell hjernesegmentering er fortsatt den beste måten for å få gode og nøyaktige resultater. Dette er en krevende prosess, det kan ta 6 til 8 timer å for å segmentere én hjerne (Eskildsen et al., 2012). Det er mange eksisterende løsninger som gjør hjernesegmentering automatisk. Noen av disse er veldige gode, men de kan slite med MRI bilder som har andre modaliteter (Kleesiek et al., 2016). I det siste tiåret så har dyp læring blitt veldig populært i forskjellig felter. Dette inkluderer felter som handler om å analysere medisinske bilder, og også oppgaver som gjør segmentering av medisinske bilder. I følge Litjens et al. (2017) så har mengden av publiserte artikler på dette feltet gått fra omtrent 0 i 2012 til over 200 i 2017. I denne masteroppgaven så vil tre forskjellige dyp lærings arkitekturer bli brukt til å gjøre hjernesegmentering. Disse arkitekturene vil også bli brukt til å gjøre leversegmentering. Dette ble gjort for å teste om disse arkitekturene også vil fungere på andre medisinske bilde problemer.

Dyp lærings arkitekturene vil bli testet på tre forskjellige datasett som inneholder MRI bilder av hjernen. Et av disse datasettene inneholder bare MRI bilder av pasienter med hjernesvulster. Arkitekturene ble testet på forskjellige datasett for å teste hvordan de vil prestere når de møter MRI bilder fra forskjellige kilder. Forskjellige eksperimenter med forskjellige sammensetninger av datasettene vil bli gjort for å se hvordan dyp lærings arkitekturene vil prestere.

Det tar lang tid å trene en dyp lærings metode. I denne masteroppgaven så vil dyp lærings arkitekturene bli testet på forskjellige maskinvare konfigurasjoner for å se hvor lang tid de tar å trene på de forskjellige oppsettene. Totalt vil 4 forskjellige konfigurasjoner bli testet med forskjellige prosessorer og grafikkprosessorer

Table of Contents

List of Tables

List of Figures

Abbreviations

CNN Convolutional neural network. ix, 2, 3, 8, 13–15, 19–22, 40, 72
CPU Central Processing Unit. i, 15, 39, 40
CRF Conditional Random Field. 33, 76
CT Computed Tomography. 3, 7, 21, 27

FM Feature Map. 14, 15, 22, 31, 32

GPU Graphics Processing Unit. i, iv, vii, 15, 23, 34, 36, 39, 40, 70, 71

KLD Kullback-Leibler divergence. 11, 28, 37, 44, 45

MRI Magnetic Resonance Imaging. i, ii, 2, 3, 5–8, 18–20, 65, 67–69, 71

RQ Research Question. 3, 66, 68, 71, 72

Chapter 1

Introduction

1.1 Motivation and problem description

Brain extraction or skull stripping is the task of producing a brain mask that marks pixels or voxels that constitutes the brain in a medical image. Voxel being an image element in 3D space unlike a pixel which is an image element in 2D space. There are many reasons for why skull stripping is an important task in medical imaging segmentation, Kleesiek et al. (2016) lists some these: cortical structure analysis (Thompson et al., 2001), cortical reconstruction (Tosun et al., 2006), thickness estimation (MacDonald et al., 2000), image registration (Klein et al., 2010) and tissue segmentation (de Boer et al., 2010). This is why there have been developed many methods that perform skull stripping over the years (Eskildsen et al., 2012). Many of these methods use complex algorithms to determine the brain mask and some of them require a lot of tuning to use. Additionally, these methods struggle when faced with tumors and other medical imaging modalities.

Deep learning is a machine learning method that solves problems in a similar manner to how a human brain solves problems. This past decade it has become a very powerful instrument for solving various tasks such as speech recognition, language processing and numerous imaging tasks. It has also opened up many possibilities for more accurate tools for tasks such as prediction, segmentation and analysis of medical images. Deep learning methods are also relatively easy to deploy and a deep learning architecture that is built for one task can be trained to work on an entirely different problem. Another advantage that deep learning methods have is that they require little tuning for them to work well. Deep learning can therefore be used to make more accurate and unbiased tools that perform skull stripping.

Skull stripping methods based on deep learning will learn how to determine the brain masks based on a training set of already segmented brain masks with little tuning required. The paper Kleesiek et al. (2016) introduces a method for skull stripping that uses a fully Convolutional neural network. Their method is compared with six existing skull stripping methods on three

different data sets. It performs better than all of the conventional methods in some metrics and Kleesiek et al. (2016) also says that their method should work well on other medical image modalities in contrast to the existing methods.

In the precursor to this master thesis (Eide, 2017) a reimplementation of the CNN from Kleesiek et al. (2016) was used to do skull stripping experiments. This architecture was tested on data from LBPA40 (Shattuck et al., 2008) and data from OASIS (Marcus et al., 2007). One research question in Eide (2017) was:

"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?"

This master thesis aims to further expand upon this research by using three different deep learning architectures and three different data sets.

In this master thesis, the CNN used in Kleesiek et al. (2016), the 3D U-Net introduced in Çiçek et al. (2016) and DeepMedic introduced in Kamnitsas et al. (2016) will be used to perform skull stripping. As mentioned earlier, the CNN was reimplemented in Eide (2017), reimplemented in Eide (2017), the 3D U-Net was reimplemented for this thesis and the DeepMedic was tested using the existing implementation[1]. The architectures will be trained and tested on MRI images from the OASIS data set, LBPA40 data set and data from the St. Olavs hospital in Trondheim, Norway.

All of the patients in the St. Olavs data set have brain tumors. The existing solutions that are not based on deep learning do not handle this well. Therefore, it would be interesting to see if deep learning can be used to tackle this problem. In Kleesiek et al. (2016) they showed that their deep learning method was better compared to existing skull stripping methods at handling images of patients with tumors in their data set. Consequently, one purpose of this thesis is to explore if deep learning methods can handle the tumor data from St. Olavs adequately.

There are Varying amounts of data in the three data sets used in this thesis. There are many more MRI images in the data from the St. Olavs hospital compared to the other two data sets. This unbalanced composition of the data sets can cause problems for deep learning methods. It can cause problems such as the deep learning methods being very good at predicting on data from the largest data set, but poorer on data sets with the fewest amount of images. Also, the MRI images in OASIS, LBPA40 and the St. Olavs data set all have different voxel sizes. This enhances the variation between the data sets causing the deep learning methods to have more complications with finding a way to relate the data from each data set with each other. Kleesiek et al. (2016) also mentioned that a significantly different resolution can have a notable impact on the predictions for their implementation. This thesis will explore how the different deep learning architectures cope with this variability in the data sets.

Deep learning methods usually take a long time to train for them to produce adequate results.

[1] https://github.com/Kamnitsask/deepmedic

Faster training usually requires more expensive hardware. In this thesis, the U-Net and the CNN will be trained on different hardware configurations to see how long their training times are. The different hardware configurations will then be assessed to get an idea why some configurations where faster than others.

Finally, the architectures will be used to do liver segmentation. The data for this experiment are Computed Tomography (CT) scans from the LiTS 2017 data set[2]. This was done to see if the architectures used for skull stripping can also be used on another medical imaging segmentation problem.

1.2 Project goals and Research Questions

The overall goal for this thesis is to further explore deep learning applied to skull stripping, especially on data from the St. Olavs hospital. Another goal for this thesis is to examine how different hardware configurations impact the training time of different CNNs. Also, this thesis will test if these deep learning methods can be used to solve liver segmentation. Based on the problem description in the previous section and the goals, the following Research Question (RQ) have been defined and will be addressed in this thesis:

RQ 1 How important is it for different deep learning architectures to train on data from the same source before performing skull stripping?

RQ 2 How important is the balance between the amount of data in each data set for deep learning architectures when doing skull stripping?

RQ 3 Is it important for deep learning architectures to have MRI scans with the same voxel size when performing skull stripping?

RQ 4 How will different hardware configurations impact the time it takes to train a deep learning method?

RQ 5 Can these deep learning architectures also be used to do liver segmentation?

1.3 Contributions

The contribution of this thesis is a comprehensive analysis of deep learning applied to skull stripping. Three different deep learning architectures were tested on various compositions of three different data sets containing MRI scans of the head. One of the data sets used in thesis contains highly variable structures, tumors, making the data set extra difficult to do skull stripping on. The experiments conducted in this thesis show that the deep learning architectures are able to produce good results on this data set, if they were first trained on the data set. Experiments also show that some of the deep learning architectures were also able to perform liver

[2] https://competitions.codalab.org/competitions/17094

segmentation on a data set containing CT scans of the abdominal region. This shows that the architectures were able to perform other segmentation tasks on other anatomical structures and on other imaging modalities.

1.4 Thesis structure

This master thesis is divided into seven chapters. The first chapter gives an introduction to the project and lists the research questions for this thesis. The second chapter gives an introduction to the theory used in this master thesis. The third chapter presents the methods and the experiments performed. In the fourth chapter the results are presented and in the fifth chapter the results are discussed in light of the background theory. The sixth chapter is the conclusion and it also presents future work that can be done to improve upon the methods used in this thesis.

Chapter 2

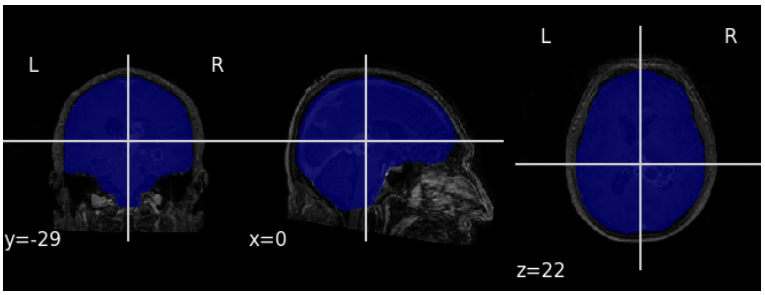
Background

This chapter introduces some of the underlying theory used in this thesis. It also contains a review on methods used for skull stripping and a review on deep learning methods that are used for analyzing medical images.

2.1 Skull stripping

Skull stripping is the process of removing non-brain tissue from a medical image of the brain, such as an MRI scan. It is a segmentation task where pixels or voxels in the image are labeled either as brain or as non-brain. As previously mentioned in the introduction skull stripping is an essential part for many medical image analysis applications that study the brain. Many of these applications rely on accurate segmentations of the brain to produce good results. Also, the skull stripped images are easier to do further analysis on for applications that are dependent on studying just the brain. A good and non-biased segmentation method for skull stripping is therefore a very valuable tool.

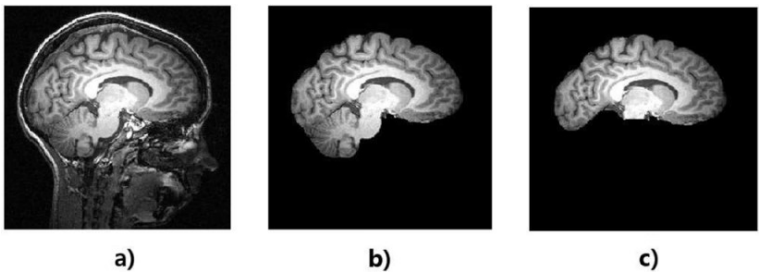
Figure 2.1 shows the coronal, sagittal and axial slices, respectively, of a MRI scan. The voxels that constitute the brain are colored in blue.



Blue marks where the brain is in the MRI scan

Figure 2.1: *Skull stripping example.*

There are some disagreements in the field on what exactly a brain mask should include and what it should not include. The two major different camps are illustrated in figure 2.2.

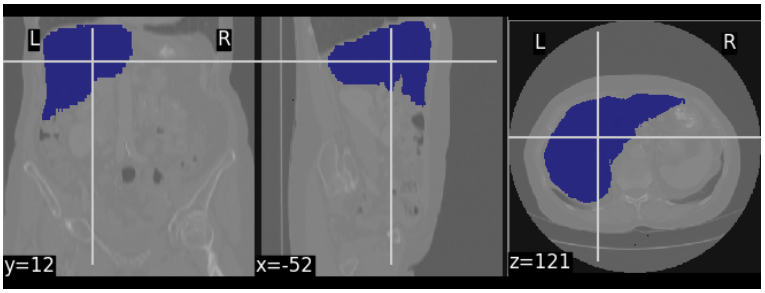


(a) *Full slice of a MRI scan.* ***(b)*** *Skull stripped brain including the cerebellum and the brainstem.* ***(c)*** *Skull stripped brain without the cerebellum and the brainstem.* Iglesias et al. (2011)

Figure 2.2: *Skull stripped brain example.*

2.2 Liver segmentation

Liver segmentation is the task of finding the liver pixels or voxels in an image. Usually the liver image is a MRI scan or CT scan. Liver segmentation can be an important pre-step for many applications such as diagnosing hepatic diseases (Ling et al., 2008) and surgical planning (Yang et al., 2017). Liver segmentation is a challenging task, especially on 3D data, because of the broad variations in liver shapes (Ling et al., 2008). Figure 2.3 shows a liver segmentation example of a CT scan.



Blue marks where the liver is in this CT scan of the abdominal region

Figure 2.3: *Liver segmentation example.*

2.3 Imaging modalities

2.3.1 MRI

Magnetic Resonance Imaging (MRI) is an imaging technique that is regularly used for taking images used for medical analysis (Preston, 2006). The images are obtained by using a strong magnetic field to align hydrogen atoms inside the body. Radio frequency energy from the machine is then used to excite the hydrogen atoms. After the machine stops emitting radio frequency energy, the hydrogen atoms return to their resting state causing the atoms to emit energy. The energy is then read by antennas inside the MRI machine. Based on the energy that the atoms emit, a grayscale image is produced. The different shades in the image correspond to different energy levels that the MRI machine has read. Different shades correlate to different tissue types in the image.

Different tissue types can be highlighted in a MRI image by varying the time between each pulse of radio frequency, i.e. the repetition time (TR), and when the energy that the atoms emit are read, i.e. the time to echo (TE). Usually the images are weighted in three different ways:

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

For analyzing the brain, MRI scans are the most used modality (Akkus et al., 2017). Though other methods like CT and PET exist, MRI images are often preferred because it offers high resolution images with a high contrast between soft tissues.

2.3.2 Computed Tomography

Computed Tomography (CT) scans are obtained by using X-Rays to build an image of a patient[1]. Conventional X-ray scans are produced by firing electrons onto a target made of a specific material. When the electrons hit the target, they release energy in the form of X-rays. The generated X-rays are targeted at a patient with a film placed on the opposite side of the X-ray generator. When the X-rays reach the film, the film is darkened. Different structures inside the patient are then highlighted on the film. Bones are white since they absorb almost all of the X-rays passed through the patient, while soft tissues are dark since almost all of the X-rays pass through.

CT scans can produce 3D images by using X-rays on different angles of the object. This is done by having X-ray beams go through the object and X-ray detectors on the opposite side of the beam. These X-ray detectors replace the film used in conventional X-Ray scans. Multiple

[1] https://www.nibib.nih.gov/science-education/science-topics/computed-tomography-ct

images from multiple angles are taken by rotating the beam and the detectors around the object. A 3D image can then be constructed by stacking the different 2D images together.

2.3.3 Storing of medical image data and image formats

There exist different ways of storing medical images. One of the most popular formats is storing them as NIfTI files. This format was created by Neuroimaging Informatics Technology Initiative (NIfTI)[2]. The files then contain the data itself as a 3D or 4D tensor and a standardized way of storing the metadata of the image including resolution, dimension and the position of the image relative to the MRI machine[3]. The information related to the position and dimensions of the image is stored in a matrix called the affine matrix.

2.4 Deep neural networks

This section describes some of the basic theory behind deep learning. A general overview of artificial neural networks is presented first and CNNs are presented later in this section. The book Goodfellow et al. (2016) presents a more thorough review on deep learning methods and their applications. The theory for this section is also based on the book Nielsen (2015).

2.4.1 Neural network

Simply explained, a neural network is a machine learning method that learns how to solve a given problem by tuning its parameters to fit the problem's underlying function. Its parameters are tuned by training it with examples of the problem and its corresponding solutions. When the network is fed examples it has not seen before it can make predictions on what the output should be. Usually neural networks have many parameters, this gives them a very a high representational capacity. In fact, a neural network can represent every existing function given that it has enough parameters. This gives them the ability to learn many of features of the problem they are supposed to solve.

An artificial neural network consists of an input layer, an output layer and most of the time it also has hidden layers. Figure 2.4 gives an example of a fully connected neural network with three layers. Each layer is connected to a layer "upstream" and a layer "downstream" by modifiable weighted connections, except the input layer which is not connected to a layer upstream and the output layer which is not connected to a layer downstream. Each layer consists of neurons that computes one output based on outputs from the neurons in the previous layer. If the network

[2] https://nifti.nimh.nih.gov/
[3] http://nipy.org/nibabel/coordinate_systems.html

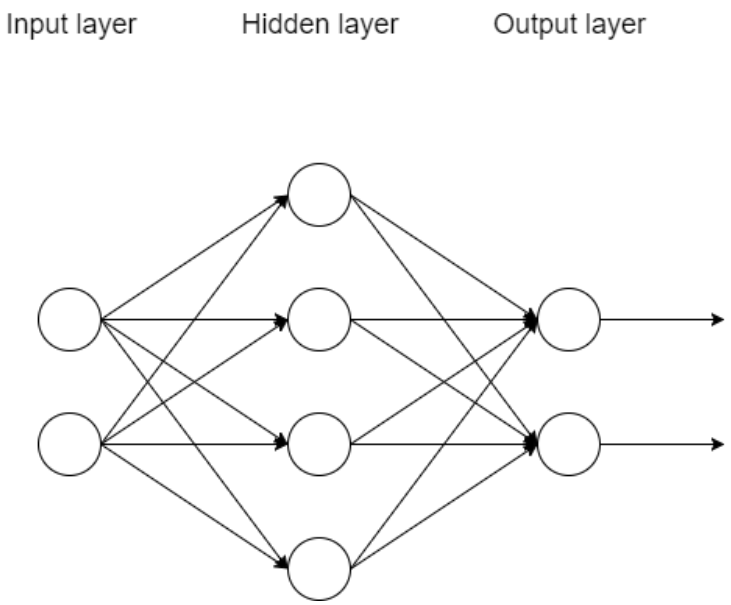


Figure 2.4: *Example of a simple neural network with three layers.*

is fully connected the neurons compute the output based on all the outputs from neurons in the previous layer.

2.4.2 Neuron

Equation 2.1 shows how a neuron computes its output based on input from a layer upstream. In the function σ is the *activation function*, w is the weights going in to the neuron from the previous layer, a is the output from a neuron in the previous layer and b is the bias of the neuron. This means that the output of a neuron is a function of the sum of all outputs from neurons in the previous layer times their weighted connections plus a bias.

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

2.4.3 Activation function

The activation function of neurons gives a restriction on the values that the neuron can output. Different activation functions can be beneficial for different tasks and they can have certain problems on other tasks.

Rectified Linear Unit

ReLU is one of the most popular activation function in use today. It is defined by this equation:

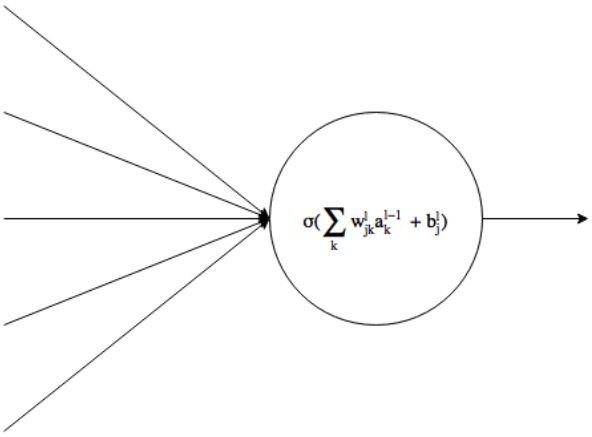


Figure 2.5: *Caption.*

$$\sigma(z) = max(0, z) \tag{2.2}$$

With this equation the neuron outputs its computed value if it is over zero, otherwise it outputs zero.

PReLU

He et al. (2015) introduced the PReLU activation function. It is defined as follows:

$$a_i x \text{ for } x \leq 0$$

$$x \text{ for } x > 0$$

Where a_i is a learned parameter which is tuned during the training of the network. If the term a_i is 0, PReLU becomes equal to ReLU.

Softmax

Softmax is an activation function often used in the last layer of a neural network. It outputs the probability of z_j given all other z. This is very useful when the network is supposed to do a classification on the inputs. The network can then output a probability vector where each value in the vector is the networks calculated probability that the input belongs to that class. The formula for softmax is given by:

$$\sigma(\vec{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1, ..., K \tag{2.3}$$

Were z_j is the output from neuron j and K is the number of neurons in the layer.

2.4.4 Loss function

The loss function measures how well a network is predicting. It is calculated based on the prediction the network makes and the corresponding truth or label for the data. Mean squared error is one of the simplest. It is given by this function:

$$MSE = \frac{1}{n}\sum_{x}(y(x) - a)^2 \tag{2.4}$$

$y(x)$ is the vector of desired labels for case x and a is the vector of the actual outputs from the network. n is the number of cases that the error is calculated for. The larger the error in the output from the network, the larger MSE is.

Kullback-Leibler divergence

The formula for Kullback-Leibler divergence (KLD) is:

$$D_{KL}(P||Q) = -\sum_{i} P(i)log(\frac{Q(i)}{P(i)}) \tag{2.5}$$

P and Q are both probability distributions. KLD measures the distance between the two probability distributions. If P and Q are equal the formula outputs 1 and if they are not equal the formula will output a value between 0 and 1.

2.4.5 Training

A neural network is trained by giving it example cases and their corresponding solutions. The network computes the prediction and the loss for the example cases. The parameters in the network are then updated using gradient descent with backpropagation. This algorithm basically computes how much each parameter contributes to the loss function. The parameters are then updated so that the loss decreases.

Each parameter's contribution to the loss function is estimated to be the gradient loss with regards to the parameter. So given parameter θ its gradient with regards to loss function C is $\frac{\partial C}{\partial \theta}$. Each parameter is updated by their negative gradient multiplied by a given learning rate: $\theta_{n+1} = \theta_n - \eta\frac{\partial C}{\partial \theta}$ where η is the learning rate. To compute all the gradients for each parameter the learning method uses an algorithm called *backpropagation*. Briefly explained, backpropagation works by computing the gradients for the output layer. These gradients are then used to compute the gradients for the previous layer. This continues until it has computed the gradients for all the parameters.

This training process where parameters are updated is continued until the network produces satisfactory results on the training data or it is stopped after a predetermined number of updates of the parameters. The amount of training a network must do is generally problem specific. The training is usually divided into *epochs*. One epoch is defined as one cycle where the network has seen all the data in the training set.

The learning rate regulates how fast a network learns. It is a value between 0 and 1. If the learning rate is too low the network will not learn fast enough. If the learning rate is too big the network may never learn the problem's underlying function and instead the loss will jump up and down and gradient descent will never find the parameters that gives the minima of the loss function. When the network is doing this, it is said that the training is diverging.

Batch training

The network can update its parameter after seeing more than one example. This is called batch training. When batch training is used the network is fed a *batch* of example cases, the loss is computed and averaged for the batch. The amount of examples fed into the network is called the *batch size*. Since the loss is averaged over a set of example cases, instead of just one example case, it becomes a more accurate estimation of how close the network's function is to the problem's underlying function. The updates to the parameters are therefore more stable. Consequently, the network can learn more smoothly. A larger learning rate can also be used when a larger batch size is used.

Batch normalization

Batch normalization works by normalizing the outputs of the previous activation layer (Ioffe and Szegedy, 2015). This reduces the networks internal covariate shift. Covariate shift means that the small changes to the networks parameters gets exaggerated in the layers downstream. By reducing the covariate shift the network can learn faster.

Optimizers

There are different alternatives to plain backpropagation for training neural networks. Sebastian (2016) provides an overview of some of the alternatives. By using momentum, the updates can be estimated to be larger or lower for the parameters for each update step. It does this by adding a fraction of the last update to the current update value. Nesterov accelerated gradient is an expansion on this. It approximates future values of the parameters to be updated. Then it uses this to update the parameters.

Optimizers can also work by adjusting the learning rate for individual parameters. One of the first optimizers that used this is called Adagrad. This optimizer makes larger updates to features

that are sparser in the data. This makes it suitable to use when the data set is small. It does this by scaling the learning rate for each parameter by the sum of its past gradients. RMSProp is another optimizer. It is similar to Adagrad, but instead of using the sum of its gradient history to scale the learning rate it scales the learning rate by a running average of the recent gradients for the parameter. Adam is one of the most recently introduced optimizers. This optimizer uses both a first order momentum and a second order momentum to scale the learning rate for each parameter.

When using optimizers, such as Adam, the initial learning rate of the network becomes a less important factor. This is because the adaptive scaling of the learning rates for each parameter is often better than setting a fixed learning rate for the entire model.

2.4.6 Convolutional neural networks

A Convolutional neural network (CNN) is a deep learning method that is very well suited for tasks related to analyzing images. CNNs are much better suited for these kinds of tasks in part because its layers are not fully connected. Because of this every neuron does not have to calculate values from every part of the image. Instead, it only needs to process parts of the image. This is favorable because not all parts of the image contains useful information when looking for a specific feature. A CNN usually has three types of layers: convolutional layers, pooling layers and fully connected layers.

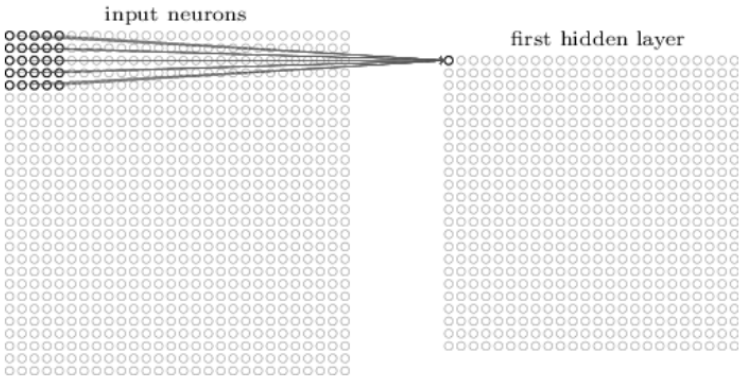


Figure 2.6: *CNN example from Nielsen (2015).*

Convolutional layers

A 2D convolutional layer computes its output given this formula called a convolution:

$$a_{j,k} = \sigma(b + \sum_{l} \sum_{m} w_{l,m} a_{j+l,k+m}) \tag{2.6}$$

σ is the activation function, b is the bias, $w_{l,m}$ is the weight and $a_{j+l,k+m}$ is the output from a previous layer. The size of l and m gives the number of neurons used to calculate a neuron in the next layer. Figure 2.6 illustrates this. In the picture the lines are the weights and the dots to the left constitutes the activations from the previous layer. The weights and bias constitute a *kernel*. l and m gives the size of the kernel and is said to be the *kernel size*.

Equation 2.6 is slided across the image according to the size of the stride. If the stride is 1 the kernel goes over the image one step at a time.

The output of this operation is called a Feature Map (FM). It is dubbed this because the kernel detects features in the image. For example if the CNN was trained to detect if an image contains a human face one kernel could be used to detect if eyes are present and another kernel could be used to detect if a mouth is present. A convolutional layer can produce many FMs by having many kernels. Consequently, CNNs can be trained to detect many different features in an image.

A convolutional layer downstream from another convolutional layer uses all of the previous layer's FMs to produce a set amount of FMs. If the input was a 2D image and the first convolutional layer outputs 3 FMs, the kernel in the next layer will be of size $l * m * 3$.

Transposed convolution

In contrast to a normal convolutional layer a transposed convolutional layer produces an output larger than its input. It does this by performing a normal convolution, but it pads its input. The padding is done so that the output size is the same as if the convolution was performed on the output it would produce an FM the same size as the input to the transposed convolutional layer. Transposed convolutional layers, in contrast to simply up sampling the FM, also contains parameters that can be changed to minimize the loss function. A more detailed explanation can be found in Dumoulin and Visin (2016) and Zeiler et al. (2010).

Pooling

Pooling layers are layers in a CNN that take each FM produced by a convolutional layer and produces a condensed version of them. One of the most used version of this is max-pooling. Max-pooling outputs the largest value in each segment of a given size in the image. The size of the segment is decided by a pool size. Stride is used to denote how the operation is slided across the image. F. ex. given a 2D max-pooling layer which has stride that equals 1 and a pool size that equals (2, 2), the max-pooling layer outputs the largest value in every (2, 2) segment of its input. If the stride was 2 the pooling operation would be moved 2 pixels to the right or 2 pixels down for each operation.

When max-pooling is used it is said that the network only keeps the features that it deems to be most important. One drawback of using pooling layers is that the CNN loses information about

the exact location of features.

Padding

To make the output of a convolution equal in size to the input the image can be padded before the convolution. Equal in size meaning the size of the FMs and not how many FMs are produced. This is also useful for making sure the convolution processes all the values in its input equally since normally the inputs at the edge of a tensor are only used for one convolution operation.

Patch training

When patch training is used the network is fed parts of the image instead of feeding it whole images. For example, if the image is $(100, 100, 100)$ the network can be fed patches of size $(20, 20, 20)$ for training and predicting. This is done because of memory constraints on the machine that the network is running on, or that it is not useful for the network to operate on the whole image.

2.4.7 Skip connections

A connection from a layer in a neural network to a layer downstream in the network, except the layer immediately downstream from the layer, is called a skip connection. In this case the layer in the deeper part has inputs from both the previous layer and a layer that is further back in the network. This can be useful for ensuring that information is kept when data is passed through the layers of a neural network. For very deep neural networks this can especially be the case.

2.4.8 GPU training

The computations done for a CNN are heavily parallelizable (Kim et al., 2017). Many computations do not have to be computed in series, but instead they can be computed in parallel. For example to compute a FM, kernel computations are done that are not dependent on one another in the same layer. Because of this GPUs are generally much faster for training CNNs compared to CPUs. CPUs are more designed to do fast processing on data in sequence while a GPU is designed to do many parallel computations at once.

2.5 Generalization, overfitting and underfitting

A network generalizes well when it makes good predictions on data it has not seen before. This is a very important measure of how well a network performs. Overfitting is when a network

makes accurate predictions on data it has already seen before, but it makes bad predictions on data it has not seen. Overfitting can be expressed as:

Two machine learning models H and H' are trained on the same training data and tested on the same testing data.

$$Error_{train}(H) < Error_{train}(H') \tag{2.7}$$

$$Error_{testing}(H) > Error_{testing}(H') \tag{2.8}$$

If equation 2.7 and 2.8 holds then H overfits on the training data compared to H' because it has a smaller error on the training data, but a larger error on the testing data.

A network underfits the data when it does a bad job at approximating the underlying function of the data. This can happen if the network is not large enough. If it is not large enough it may not have the representational capacity to express the problem's underlying function. Underfitting can also happen if the network is not trained long enough on the data.

Overfitting can occur when a model does not have enough data to train on. The network can then learn to represent the data as well as noise contained in the data. Good methods for preventing this kind of behavior in machine learning exists and some of the methods are discussed in the next subsection.

2.5.1 Regularization

Regularization techniques are a set of methods developed to avoid overfitting and produce models that generalize better to the problem's underlying function.

L1 and L2 regularization

These regularization techniques work by adding to the cost function a value determined by how large the weights in the network are.

L1:

$$C = C_0 + \frac{\lambda}{n} \sum_w |w| \tag{2.9}$$

L2:

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2 \tag{2.10}$$

λ here is the regularization parameter. L1 and L2 regularization works by making the network learn smaller weights. With smaller weights the behavior of the network will be less volatile to small changes in the input. This makes the network better at handling noise that the data may have.

Dropout

Dropout is a regularization technique that works by deleting a percentage of the neurons in a network in a given layer. The percentage of deleted neurons is called the dropout rate. The input is then propagated through the network and the weights are updated. In the next update step the deleted neurons are restored and a new set of neurons are deleted. The process of updating the weights based on a new input is then repeated. These steps are repeated for as long as the network is training.

When the network is used for predicting, the weights are scaled down according to how large the dropout rate is. This is done because the weights are tuned to the problem when less weights are present.

Dropout works because, essentially, it trains many different networks with different neurons. It then uses the average of these networks to do predictions. The "individual networks" may then have overfitted different parts of the data, but most likely the averaging works so that the final network does not overfit the data.

Artificially expanding the training data

Artificially expanding the training data or data augmentation works by modifying the data a little, but the modification must be so that it still is possible to classify the data the same in the real world. This can improve the machine learning method's ability to classify inputs partially because the available amount of data to train on becomes greater. If the machine learning method is supposed to classify images, the images could be rotated. For speech, background noise could be added. It is however important that the changes to the data mimics real world modifications that can happen to the data.

Expanding the training data also has a regularizing effect on the model because the model should be more robust to changes in the data and it should be more difficult for the model to overfit the training data because there is much more of it.

2.6 Training, test and validation data

To avoid a model overfitting the data and to get good generalization the available data is often split into three sets. The training data is used for training the model and the test data is used to

score how well the final model is performing. Validation data is used for tuning the hyperparameters. It can also be used to determine when to stop the training of a neural network by for example, ending the training when the error on the validation data is under a threshold. When a model has a low error on the training data, but a high error on the test data, the model is said to overfit the training data.

2.7 Literature study

In this section previous work that has been done on medical imaging segmentation will be described. This section will first provide an overview of previous methods used for skull stripping, then a general exploration on where deep learning has been used for analyzing brain images. The third part will look at deep learning methods that have been used for liver segmentation. The last section will describe a novel deep learning architecture called capsule networks.

2.7.1 Previous methods used for skull stripping

Many programs that perform skull stripping have been developed. According to Kleesiek et al. (2016) some of the most popular ones include:

- BET (Smith, 2002)
- 3DSkullstrip
- Hybrid watershed algorithm (HWA) (Ségonne et al., 2004)
- Brain surface extractor (BSE) (Perona and Malik, 1990)
- ROBEX (Iglesias et al., 2011)
- BEaST (Eskildsen et al., 2012)

BET is one of the most widely used since it is a quick and easy skull stripping tool to use. The solution works by having a deformable model at the center of gravity for the image. The deformable model is then grown until it fits the supposed borders of the brain. This program works most of the time, but it can struggle when it is faced with tumors because it hinders the growth of the deformable model (Speier et al., 2011). 3DSkullStrip is similar to BET, but it also uses points outside the center of gravity to grow the deformable model. 3DSkullStrip is part of AFNI which is a tool for analyzing MRI images (COX, 1996). HWA uses a watershed algorithm combined with a deformable surface model to create the brain mask. BSE uses a region proposal algorithm to find the brain mask. ROBEX uses random forests and a point distribution model. BEaST uses a library of data and the sum of squared differences to estimate the brain mask. BEaST will perform better if its library contains data from the data set it is supposed to do skull stripping on.

All of these methods reportedly perform unsatisfactory when faced with brain tumors (Speier et al. (2011), Kleesiek et al. (2016), Eskildsen et al. (2012)). What's more is that all of them,

except BEaST and ROBEX, require the user to tune parameters to get a satisfactory result. This can cause problems since the brain masks are often used for further analysis. Unreliable tools can hinder the reproducibility of these analyses. Further, all these methods, except BET, were developed to only work on T1-weighted scans.

2.7.2 Deep learning architectures used for analyzing brain images

According to Ker et al. (2018) the advance of deep learning has produced many benefits for the medical field. They also state that it has the potential to alter how the field is practiced in the coming years. This is also backed by Litjens et al. (2017) who also remarked that the amount of papers published on this subject has exploded in the recent years. Ker et al. (2018) also mentions several papers that apply deep learning to medical imaging applications. Of these, there are many papers that cover segmentation of medical images and most are concerned with doing segmentation on images of the brain. Ker et al. (2018) also recommends the review paper by Akkus et al. (2017) which provides a survey on different architectures used for analyzing brain MRI scans. According to Akkus et al. (2017) some of the most popular types of CNN architectures used for brain MRI segmentations are:

- Patch-Wise CNN Architectures
- Semantic-Wise CNN Architectures
- Cascaded CNN Architectures

The patch-wise approach feeds the architecture with patches of the image and makes predictions on them. This is currently the most popular approach for analyzing brain MRI images. A twist on this is using multiple inputs for the patches where each patch is processed by a different pathway of the CNN. The CNN's pathways process patches of different size, but the patches are used for predicting the same pixels or voxels. These types of CNNs are called multiscale CNNs. Cascaded CNN architectures are two CNNs combined where one is used for predictions and one is used to tweak the results to make them better. Semantic-wise CNN predicts a class for each pixel or voxel on the whole input image. The architectures include one encoder part and one decoder part. The encoder part uses convolutions and pooling to extract features from the image. The decoder part uses upsampling and transposed convolutions on the higher-level features from the encoder part. The encoder and the decoder path can be connected with skip connections so that the decoder part can also extract lower level features from the encoder part.

Akkus et al. (2017) also divide the architectures based on which problems they solve. One for *segmentation of normal brain structure* and one for *segmentation of brain lesions*. Most of the architectures presented for the former are 2D CNNs with patch-wise predictions. One architecture used a semantic-wise CNN architecture (Nie et al., 2016). The architectures used for segmenting brain lesions were also mostly patch-wise. Six architectures were 2D CNNs and three architectures were 3D CNNs. One of the proposed architectures is Zhao and Jia (2016) which predicts brain tumors based on 2D slices of sagittal, coronal and axial planes of MRI

scans. The network is trained on 2D images from the different planes. When the network is used for segmentation it predicts 2D images which are then combined to make a 3D image of the proposed tumor location. Because the architecture is a 2D CNN the network is much faster compared to a 3D CNN, since 3D CNNs are generally much more resource heavy to run. Their results were comparable to other machine learning algorithms on BRATS 2013, which is a brain tumor segmentation challenge. Kamnitsas et al. (2017) presents a multiscale CNN. The CNN uses two pathways where one stream of the architecture operates on a normal resolution patch, while the other stream operates on a lower resolution patch. The architecture had the top-ranking performance on BRATS 2015.

In the end, Akkus et al. (2017) state that even though deep learning is relatively new in its usage on MRI scans it outperforms most of the conventional machine learning methods that are used. This is explained by deep learning architectures' ability to handle complex structures without much manual tuning. Some of the challenges ahead for deep learning applications in the medical imaging field is the availability of data that the networks can train on. This can be somewhat mitigated by using data augmentation. Transfer learning is also proposed. This is when a network is first trained on data that is not related to the data it is going to predict on. Afterwards the network is trained on the data which it is supposed to predict on. This can help the network do better predictions because the network's parameters are more tuned for doing predictions when the network starts training on the real data.

Ker et al. (2018) also mentions some CNN architectures used for MRI segmentations. One of them is an architecture called V-Net proposed by Milletari et al. (2016) that is inspired by U-Net introduced in Ronneberger et al. (2015). This architecture is a 3D CNN compared to the original U-Net which is a 2D CNN. The architecture was used to segment MRI prostate images and got a score similar to other top scoring architectures on the data set. Milletari et al. (2016) also introduced a new type of loss function based on the dice score. The architecture is trained by trying to maximize the dice score for its predictions.

Ker et al. (2018) also mentions a study by Casamitjana et al. (2016). Casamitjana et al. (2016) compare three different 3D CNN architectures. One architecture is a single scale 11 layer CNN inspired by VGG-16 (Simonyan and Zisserman, 2014) with skip connections. The other architecture is inspired by U-Net. It uses 3D convolutions instead of 2D. The third and final architecture is inspired by DeepMedic and uses a multi scale CNN that aims to gather both the high and the low-resolution features of the image. All architectures were designed so that they could do full predictions in one forward pass during test-time. The architectures were tested on the BRATS 2015 data set. The architecture inspired by DeepMedic performed best.

Ker et al. (2018) also lists another architecture inspired the original U-Net. The implementation is presented in a paper by Çiçek et al. (2016). Like the original 2D U-Net this implementation also has one path that uses max-pooling and one part that uses up convolutions. These are respectively called the *analysis path* and the *synthesis path*. The architecture works by doing full processing on the images in one forward pass. The implementation was tested on Xenopus

kidney data. The paper states that they achieved accurate results on the data, even though the data was very variable.

2.7.3 Deep learning for liver segmentation

SLIVER07[4] is a liver segmentation challenge that has been running since 2007. Before 2016 the entries were mostly traditional image analysis methods, but in recent years deep learning methods have also placed high in the rankings (Litjens et al., 2017). The top spot in SLIVER07, however, still seems to be traditional image analysis methods[5].

Litjens et al. (2017) presents some methods used for liver segmentation. Most of the methods are 2D CNNs. Ben-Cohen et al. (2016) presents a 2D CNN based on the CNN presented in Simonyan and Zisserman (2014) also known as the VGG-16 architecture. Some modifications are made to the network such as discarding the final classification layer and using convolutions instead of fully connected layers. Also, the architecture is appended with two up-sampling layers so that the prediction size is equal to the input size. Before each up-sampling layer skip connections are used to connect to pooling layers upstream in the architecture. The input to the architecture are axial slices of CT images. The input images are modified to be three CT slices, one slice above and below the input image, where the slices were interpolated using linear interpolation. According to Litjens et al. (2017) the architecture received good results on SLIVER07.

Another architecture mentioned in Litjens et al. (2017) is an architecture presented in Lu et al. (2017). This architecture is a 3D CNN with 11 convolutional layers with mostly small kernel sizes, two pooling layers and three upsampling layers. After the first convolutional layer a local response normalization scheme is used. All layers use the ReLU activation function except the last layer where a log-regression activation function is used. This activation function estimates the probability of a voxel being part of a class. The prediction of the CNN is refined using graph cut based segmentation. Lu et al. (2017) say that the method "demonstrated superior segmentation accuracy" compared to other liver segmentation methods. According to Litjens et al. (2017) this implementation achieved "competitive results" on SLIVER07.

2.7.4 Capsule networks

A new type of deep learning architecture used for analyzing images was introduced by Sabour et al. (2017), it is called capsule networks. The type of network was an idea by Geoffrey Hinton who was also the first to introduce backpropagation in deep learning. Capsule networks try to solve problems CNNs' have when they are faced with rotations of images that they have not trained on yet. Capsule networks are based on the idea that the human brain contains something

[4] http://www.sliver07.org/
[5] http://www.sliver07.org/results.php

called capsules which are responsible for handling detection of objects with different rotations, size, position, shape etc.

In a capsule network neurons are replaced by "capsules". The capsule will output a vector instead of the scalar that a neuron outputs. The length of the output vector represents the probability that a given object exists. The capsules are used to compose layers, where much like a CNN, the lower layers try to detect lower layer features and capsules in the higher levels detect more complex features. In the paper the first capsule layer is composed of two convolutional layers which output 256 FMs. The outputs of the CNN are then squashed to output vectors instead of scalars. The paper also introduces an algorithm called routing by agreement. In short, the algorithm is used for sending the output from capsules in the lower level to a specific capsule in a higher level based on the likelihood that it belongs to the class that the capsule is supposed to detect.

The network was tested on MNIST data, which is a data set of handwritten digits, and it achieved state of the art performance. It was also tested on MultiMNIST, which contains overlapping digits, where it achieved superior results compared to existing deep learning architectures.

LaLonde and Bagci (2018) presents a use of capsule networks for object segmentation. The implementation has 95 % less parameters compared to 2D U-Net and has a slightly higher DICE score on segmentation of pathological lungs from the LUNA16 subset of the LIDC-IDRI database compared to existing architectures such as U-Net.

Chapter 3

Method

This chapter presents the chosen methods to answer the research questions. First an explanation of the different data sets is presented. This is followed by a presentation of the three deep learning architectures used in this thesis. Next the data preprocessing steps are presented followed by a description of how a model is trained using multiple GPUs. In section five the different evaluation metrics are described. The final section details how the experiments were conducted in this thesis.

3.1 Data sets

This section first describes the three different data sets used for training and testing the different skull stripping methods used in this thesis. The data sets are: OASIS, LBPA40 and data provided by the St. Olavs hospital. Lastly, the LiTS data set, which is used for liver segmentation, is presented.

3.1.1 OASIS

The Open Access Series of Imaging Studies (OASIS) is a project for making medical images of the brain freely available. In this thesis the "OASIS-1: Cross-sectional MRI Data in Young, Middle Aged, Nondemented and Demented Older Adults" data set was used. The images contained in the OASIS data set have been bias field corrected (Buckner et al., 2004). The brain masks were produced by using hidden Markov fields (Zhang et al., 2001) and then verified by experts. Therefore, the brain masks may not be perfect.

The OASIS scans are T1-weighted and contain people of ages between 18 and 96. The data set also contains scans from people who have Alzheimer's disease in early stages. Brain voxels in

the brain mask are assigned values between 1 and 4, and non-brain voxels are either 0 or 5. The brain masks include both the cerebellum and the brainstem.

For these experiments disk 1 and disk 2 in the OASIS data set were used instead of using the whole data set. This was done in part to not have an overweight of OASIS data and also so that the results were more comparable to the results in Kleesiek et al. (2016). They also only used the first two disks.

Table 3.1: *Dimension and voxel size for the OASIS data set.*

Dimensions	Vox. size	Count
176x208x176	1x1x1	76

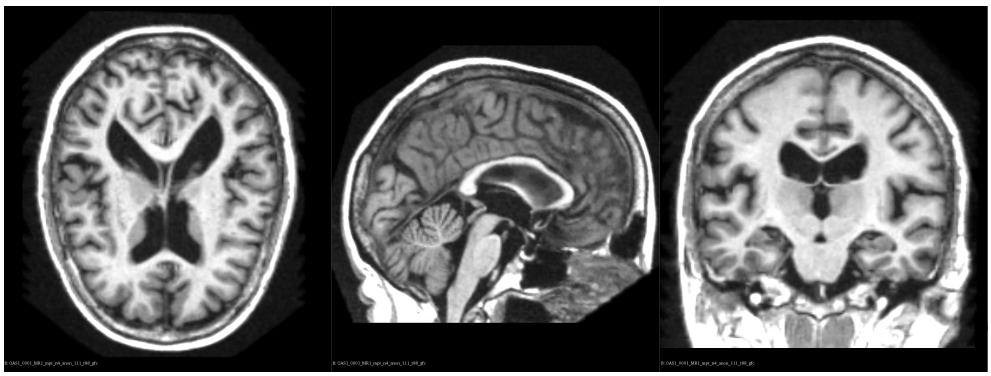


Image OAS1_0001_MR1_mpr_n4_anon_111_t88_masked_gfc_fseg from the OASIS data set

Figure 3.1: *OASIS data example.*

3.1.2 LBPA40

The LONI Probabilistic Brain Atlas (LBPA40) data set contains 40 T1-weighted scans (Shattuck et al., 2008). The data set also includes the corresponding brain mask which includes both the cerebellum and the brainstem. The brain masks were produced manually and the brain tissue voxels have been assigned value 255 and non-brain voxels are assigned value 0.

Table 3.2: *Dimensions and voxel size for the LBPA40 data set.*

Dimensions	Vox. size	Count
256x124x256	0.859375x1.5x0.859375	38
256x120x256	0.859375x1.5x0.859375	2

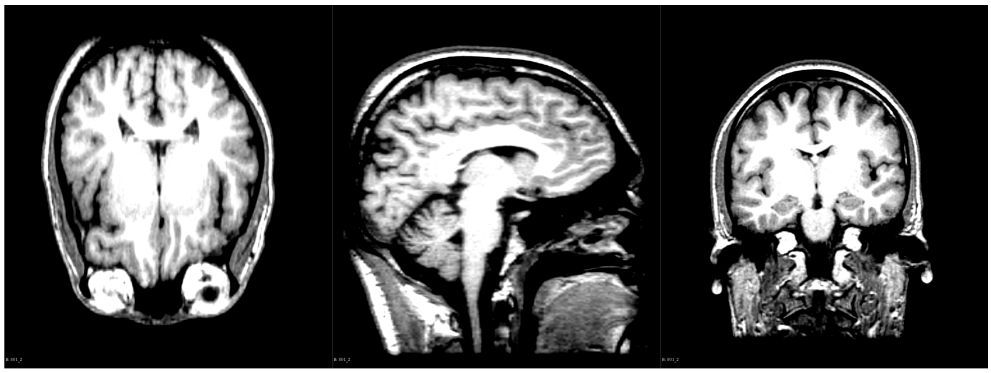


Figure 3.2: *S01.native.mri from LBPA40 data set.*

3.1.3 St. Olavs data

The data from St. Olavs contains 131 T1-weighted scans and their corresponding brain masks. In contrast to the other data sets the St. Olavs data set varies considerably in voxel size and dimensions. Also, every patient in the data set have brain tumors. This makes the data even more variable as the tumors can vary considerably in size and they can be found in different places in the brain. In figure 3.3 a tumor can be seen center-left in the axial slice. Table 3.3 shows the different dimensions and voxel sizes for the data set.

Table 3.3: *Dimensions and voxel sizes for the St. Olavs data.*

Dimensions	Vox. size	Count
496, 512, 176	0.5x0.5x1	18
256, 256, 192	1x1x1	42
512, 512, 176	0.5x0.5x1	10
512, 512, 160	0.5x0.5x1	8
256, 256, 160	0.97656x0.97656x0.99999	1
256, 256, 104	0.85940x0.85940x1.80071	1
256, 256, 180	1x1x1	16
256, 256, 189	1x1x1	1
512, 512, 167	0.5x0.5x1	1
256, 224, 128	1x1x1.33	1
512, 512, 160	0.5x0.5x1.15	2
256, 256, 184	1x1x1]	1
256, 256, 186	1x1x1.00006	1
496, 512, 164	0.5x0.5x1	1
512, 512, 133	0.5x0.5x1.1	1
512, 512, 170	0.5x0.5x1.1	1
496, 512, 168	0.5x0.5x1	1
256, 256, 174	1x1x0.99948	1
512, 512, 164	0.5x0.5x1	1
256, 256, 180	0.93750x0.93750x1	5
384, 512, 128	0.5x0.5x1.33	9
256, 256, 180	0.93750x0.93750x0.99999	1
256, 256, 180	0.93750x0.93750x1.00001	1
256, 256, 128	0.93750x0.93750x1.33000	1
256, 256, 160	0.97656x0.97656x1	2
512, 512, 192	0.5x0.5x1	1
256, 256, 150	0.97656x0.97656x1	1
480, 512, 176	0.5x0.5x1	1

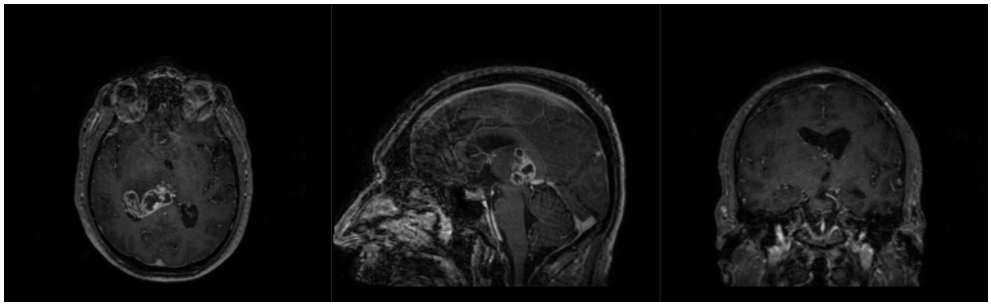


Figure 3.3: *3/coreg_T1 from St. Olavs data set.*

3.1.4 LiTS data

LiTS 2017[1] data set contains 3D CT images of the abdominal region. The data set was first used for a medical image segmentation challenge and organized for the conferences ISBI 2017[2] and MICCAI 2017[3]. The abdominal images contain a corresponding label image with three classes: background, liver and liver tumors. In this thesis the tumors were ignored, and they were instead classified as liver voxels.

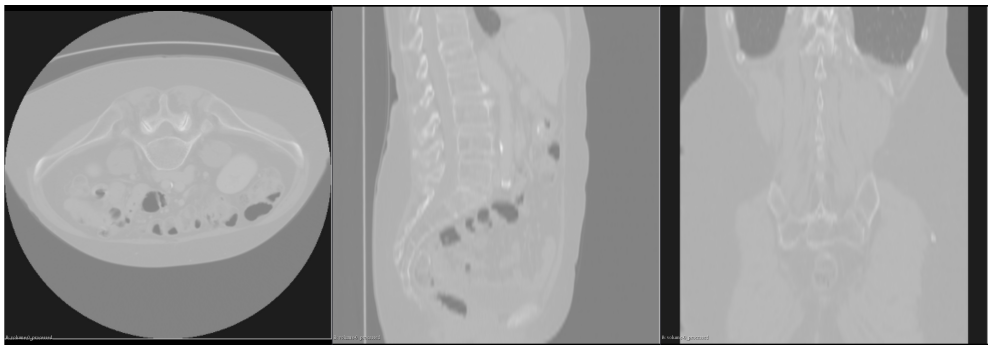


Figure 3.4: *volume-0 from LITS data set.*

3.2 Deep learning architectures

Three different deep learning architectures were used in this thesis. These architectures were chosen based on the literature review and how they differ in their implementation details. DeepMedic was chosen because of its patch based multi scale architecture. The 3D U-net was chosen because of its encoder-decoder architecture. The CNN from Kleesiek et al. (2016) was chosen because of its fully convolutional patch based approach and also because it had already been reimplemented in the precursor to this master thesis (Eide, 2017). The CNN from Kleesiek et al. (2016) will be referred to as "the CNN" for the rest of this thesis.

3.2.1 3D CNN

Kleesiek et al. (2016) introduced a fully convolutional neural network used for skull stripping. Their architecture has 11 layers with a max-pooling layer after the first convolutional layer. The implementation uses a patch based approach for training and predicting. Table 3.4 shows the implementation details. The architecture has a receptive field of size 53^3. This means that for every patch of size 53^3 it predicts 1^3 voxel. The number of predicted voxels (P) as a function of the patch size n is expressed in this equation:

[1] https://competitions.codalab.org/competitions/17094
[2] http://biomedicalimaging.org/2017/challenges/
[3] http://www.miccai2017.org/satellite-events

$$P = \frac{n - 53}{2} + 1 \tag{3.1}$$

Table 3.4: *CNN architecture from Kleesiek et al. (2016).*

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

Their architecture was evaluated on three different data sets: OASIS, LBPA40 and IBSR (Rohlfing, 2012). The results for the experiment where the CNN was evaluated using 2-fold cross validation on all the data as one pool can be seen in table 3.5.

Table 3.5: *Results for Kleesiek et al. (2016) using 2-fold cross validation.*

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

Training

The network is trained by feeding it random patches of data. In Eide (2017) patches of size 59^3 and a mini batch size of 4 was used. A corresponding label for the patch of size 4^3 is computed. This label are the central voxels for the patch that is fed into the network. The network starts with 0.00001 as learning rate. The learning rate is decreased if the loss has not been lowered in over 5000 update steps and the learning rate has not been decreased in 4000 steps. The training is ended after the learning rate has been decreased 10 times.

Before each batch is fed into the network the implementation adds a random value between $[-0, 05, 0.05]$ and then multiplies it with a random value between $[0.85, 1.3]$. This is done so that the implementation focuses less on the values themselves, but more on how the values differ compared to each other.

The network uses the ReLU activation for every layer, except the last layer where the softmax activation function is used. For training the network uses the Adam optimizer and the KLD as loss function.

Prediction

When predicting the network is fed patches of size 84^3. The network predicts a tensor of size 16^3 per patch. Using a technique called max-fragment-pooling Masci et al. (2013) the implementation's prediction is used to predict 32^3 voxels of the brain mask. Before the image is segmented into patches the whole image is padded with grey values. This is done so that the network can predict the voxels on the edges of the image.

The brain mask is built by combining all the predicted voxels. The prediction is stripped of predicted positive voxels that are not part of the largest components of predicted positive voxels. Positive voxels here means predicted brain-tissue or liver-tissue. Finally, all voxels with a value less than the threshold 0.5 are set to 0 and the rest of the voxels are set to 1.

3.2.2 U-Net

U-Net is a 2D CNN introduced by Ronneberger et al. (2015). The network is characterized by having normal convolutional layers and max-pooling layers followed by an equal number of up convolutional layers. The convolutional layers and the up convolutional layers are connected with skip connections.

The architecture was changed by Çiçek et al. (2016) to account for 3D images. This was done by using 3 dimensional kernels instead of 2 dimensional. The architecture was tested on "complex, highly variable 3D structure, the *Xenopus* kidney". Because of the up convolutional layers the architecture produces an output that has equal size input and output. This means that the architecture can train and predict on whole images. When the network processes 3D images it requires a large amount of memory to train and predict. Therefore, this implementation trains and predict using patches.

The path that uses max-pooling is dubbed the *analysis path*. This path does two convolutions per max-pooling operation. The kernel size for all convolutions are 3^3, the stride is 1 and the data is also padded before the convolutions so that the output is equal in size to the input. Figure 3.5 illustrates how many filters each convolution produces with the number on top of each blue box. In this implementation the data has only one channel, therefore the first number is supposed to be 1, not 3. In total 6 convolutions and 3 max-pooling operations are used. The max-pooling layers have a stride of 2 and a pool size of 2^2.

The other part that uses up-convolutions is called the *synthesis part*. This part consists of 6 convolutional layers and 3 transposed convolutional layers. After each transposed convolution the data is concatenated with data of equal size from the analysis path, also called skip connections. This is represented with the green arrows in figure 3.5. The skip connections enable the synthesis part to observe higher resolution features of the image. The kernels in the synthesis part are the same size as the kernels in the analysis part. Padding is also utilized in the same

way. The last convolutional layer uses a 1^3 kernel and produces two feature maps for the two classes it is supposed to predict.

Every convolution in this implementation uses the ReLU activation function except the last layer where the softmax activation function is used.

3.2.3 Training

The model is trained by feeding it random patches from the data using a batch size of 4. The training for this implementation of U-Net is equivalent to how the CNN presented in the previous section is trained. The input is processed the same way and the training is ended following the same rules. The two implementations also use the same optimizer, the only difference is that the initial learning rate was set to 0.0005 instead of 0.00001.

3.2.4 Prediction

For predictions the implementation first calculates where the patches it should predict are located according to their sizes and how much the patches should overlap with each other in the 3 dimensions. The overlap is set as half the size of the input patch. This way of doing the predictions was found on the GitHub page https://github.com/ellisdg/3DUnetCNN and their approach was rewritten for use in this implementation.

The pseudocode in algorithm 1 shows how the indices for the patches are computed. The computed indices are the starting points for the patches relative to the data tensor.

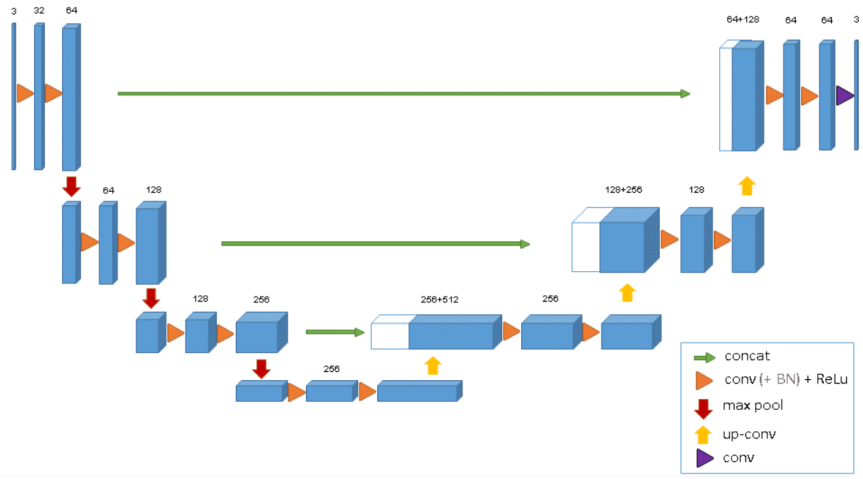
Algorithm 1 Compute patch indices

procedure COMPUTEPATCHINDICES($data\_shape, patch\_shape, overflow$)
    $n\_patches\_in\_each\_dim \leftarrow ceiling(data\_shape/(patch\_size - overlap))$
    $overflow \leftarrow (patch\_size - overlap) * n\_patches\_in\_each\_dim - data\_shape + overlap$
    $start \leftarrow -ceiling(overflow/2)$
    $step \leftarrow patch\_size - overlap$
    $patch\_indices \leftarrow \{\}$
    for $i = 0$ to $n\_patches\_in\_each\_dim[0]$ **do**
        for $j = 0$ to $n\_patches\_in\_each\_dim[1]$ **do**
            for $k = 0$ to $n\_patches\_in\_each\_dim[2]$ **do**
                $patch\_indices.add([start[0] + step[0] * i, start[1] + step[1] * j,$
                $start[2] + step[2] * k])$
    return $patch\_indices$

Patches that have overflowed the size of the image are padded with average values from the image before prediction to ensure that they are of the correct size. When the model has predicted

all of the patches the prediction is built by combining all the predicted patches. Patches that are out of bounds with the data shape are clipped so that the predicted image is of the appropriate size. When the predictions overlap the final prediction uses the average of the overlapping parts of the predictions.

The brain mask is produced by keeping every value above a threshold of 0.5 and setting all other values to 0 in the tensor.



Çiçek et al. (2016)

Figure 3.5: *Figure depicting the U-Net architecture.*

3.2.5 DeepMedic

DeepMedic is an 11-layers deep 3D CNN introduced in Kamnitsas et al. (2017). The network uses two convolutional pathways where one pathway processes a sample that is downscaled by a factor of 3 and the other pathway processes a normal resolution sample. Both samples are centered around the same voxel. The downscaled sample contains a larger context of the image compared to the normal resolution sample as can be seen with the blue lines for the lower resolution sample and the green line for the normal resolution sample in figure 3.6.

The configuration for DeepMedic used in this thesis uses a patch size of 25^3 for the normal resolution sample and 19^3 patch size for the downscaled sample. The dual pathways make it possible for the network to have a larger context for the sample it trains on while keeping the computational cost low. The number of FMs and the size of the FMs can be seen in figure 3.6 as numbers under each box where the first number is the number of FMs and the second number is the size of the FM. The first box shows the input where four channels are used, in this implementation only one channel was used for all the input data. Note also the "Up" in the pathway that processes the low-resolution image. This represents an upsampling that is done on the input. All the kernels are of size 3^3 in the two pathways. The final voxel classification is

done by concatenating the pathways and processing them first through a layer with 3^3 kernels with padding to keep the FMs the same size, and then through two layers with 1^3 kernels. The number of FMs in the last layer is dependent on how many classes it is supposed to predict. For this task the final layer produces 2 FMs in the final layer because this implementation only classifies two classes. Each layer uses the PReLU activation function, except for the last layer which uses the softmax activation function.

DeepMedic was improved in Kamnitsas et al. (2016) by introducing residual connections between every second layer. Figure 3.6 shows this as $\bigoplus$.

The implementation of DeepMedic offers an easy set up for using it on other problems. A few configuration files have to be edited in order to use it. The configuration files specify the name of the session, some of the hyperparameters for the model and what training, validation and testing data the model is going to use.

Training

The training is divided into 35 epochs. Each epoch is comprised of 20 subepochs. For each subepoch a maximum of 50 cases are loaded from the data set. 1000 patches of size 25^3 are extracted from this set. The patches are sampled from the image so that there is a 50% chance that the central predicted voxels of the patch are foreground, i.e. brain, liver, or background. The batch size of the model is 10. This means that the weights of the model is updated 100 times per subepoch. For validation, this implementation extracts n patches of size 17^3 with a batch size of 48. To ensure that the network is trained on every parts of the image the whole image is padded before it is processed.

The model starts with an initial learning rate of 0.001 and uses the RMSProp as optimizer. The learning rate is halved at predefined epochs 12, 16, 19, 22 and 25. The parameters for RMSProp is $\rho_{RMS} = 0.9$ and $\epsilon_{RMS} = 10^{-4}$. The network also uses the Nesterov momentum with a momentum value of 0.6.

The network uses both L1 and L2 regularization with $L1_{reg} = 0.000001$ and $L2_{reg} = 0.0001$. Further the network uses dropout in the last two layers for further regularization. The dropout rate is set to 50%.

The weights of the model are initialized according to He et al. (2015). Specifically the weights are initialized by sampling from a normal distribution with a standard deviation of $\sqrt{\frac{2}{n_l^{in}}}$. Where:

$$n_l^{in} = C_{l-1} \prod_{i=x,y,z} k_l^{(i)} \tag{3.2}$$

C is the number of FM in layer $l-1$. k is the size of the kernel in dimension (x, y, z). The

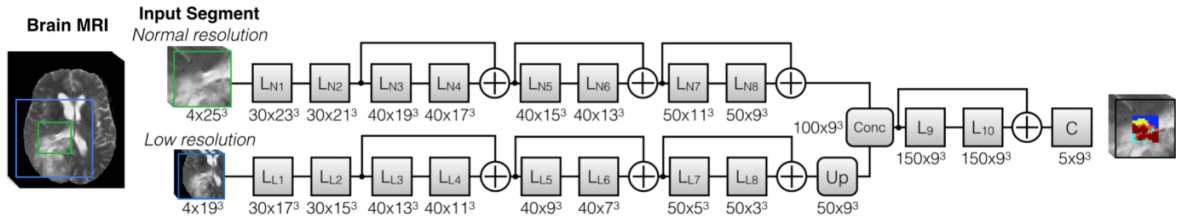
formula calculates the number of weights from the input that a neuron from layer l is connected to.

Batch normalization is used. The average of the standard deviation for the normalization is calculated over the last 60 processed batches.

Predicting

Since there are only convolutions in DeepMedic, the number of predicted voxels per patch is small. The receptive field of the architecture is 17^3 so for an input patch of size 25^3 the network will predict 9^3 voxels. For predicting, the input images are also padded.

In the implementation of Kamnitsas et al. (2016) Conditional Random Field (CRF) (Zhao et al., 2016) was used for post-processing the prediction. This was not used in this implementation, since the purpose of this thesis is the study of the different deep learning architectures and how they predict.



Kamnitsas et al. (2016)

Figure 3.6: *Figure depicting the DeepMedic architecture.*

3.3 Data preprocessing

3.3.1 Resampling

The St. Olavs, LBPA40 and the LiTS data were resampled to voxel size $(1mm, 1mm, 1mm)$ using continuous interpolation for the scans and linear interpolation for the labels. Interpolation was used to fill gaps that arise in the images during the resampling. The scans used continuous interpolation because the values are real numbers and the labels used linear interpolation because the labels are whole numbers. The resampling was done with Nilearn using the function image.resample_img[4] from the python module Nilearn. The function resamples the image to a target affine, so the image will not have its original affine anymore.

[4] http://nilearn.github.io/modules/generated/nilearn.image.resample_img.html

3.3.2 Normalization

All data will be pre-processed in the same way for all models. The images are normalized according to this equation:

$$I = \frac{I - mean(I)}{4std(I)} \tag{3.3}$$

This was also the same normalization scheme used in Kleesiek et al. (2016). The corresponding labels are also processed so that every voxel is either 1 or 0 where 1 is brain or liver tissue and 0 is non-brain or non-liver tissue. This was done because the OASIS and the LBPA40 data sets had assigned different values for brain voxels and the LiTS data set had values for both liver and liver tumors.

3.4 Multi GPU model

Keras with a TensorFlow backend supports the training of a model on multiple GPUs[5]. This is done by copying the model on the specified number of GPUs. During training or prediction, the input batch will be split and then processed on different GPUs. F. ex. with a batch size of 4 and 2 GPUs, each GPU will process 2 samples. This means that a multi GPU model will not work for batch sizes smaller than the number of GPUs specified.

3.5 Evaluation metrics

3.5.1 Dice score

The predicted voxels can be classified into these four categories:

- TP (True positive) correctly predicted voxels
- FP (False positive) falsely predicted positive voxels
- TN (True negative) correctly predicted negative voxels
- FN (False negative) falsely predicted negative voxels

F. ex. in a predicted brain mask TP voxels mean that the model has correctly predicted brain tissue, with FP voxels the model has predicted the voxel to be brain while it should have been non-brain, TN voxels are voxels that the model has predicted to be non-brain correctly and FN voxels means that the model has predicted non-brain voxels when the voxel should have been classified as brain.

[5] https://keras.io/utils/#multi_gpu_model

To evaluate the performance of the different models, three different evaluation metrics are used:

$$DiceScore = \frac{2TP}{2TP + FP + FN} \tag{3.4}$$

$$Specificity = \frac{TN}{TN + FP} \tag{3.5}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{3.6}$$

Sensitivity measures the proportion of true positives that was predicted correctly, and specificity measures the proportion of true negatives that was predicted correctly. The Dice score gives a score on how well the model predicts overall.

3.5.2 Error maps

Error maps are shown for one experiment. These will give an indication of where the architectures are predicting badly. The error maps were plotted using nilearn.plotting.plot_stat_map. Every image was first resampled to the same size using nilearn.image.resample_to_img where the first image read was used as the target image for the resampled image's dimensions. Linear interpolation was used to fill in any gaps. The error map was produced by counting every wrongly predicted voxel for every image. The error maps will not give a perfect picture of where the architectures predict wrongly, partly because the images are positioned differently in 3D space. Despite this, the error maps will give an indication of where the different architectures' predictions are bad.

3.5.3 Other metrics

The implementation of U-Net and the CNN logs loss value and accuracy on the training data as well as on the validation data. The logger also logs the time between each update step. DeepMedic logs training and validation accuracy during training.

For some of the experiments box plots will also be shown. The box plots were made with matplotlib[6]. The median score is shown with an orange line in the box and the box extends from the lower to the upper quartile of the scores. Outliers are shown as dots and the lines extending from the box shows the range of the scores.

[6] https://matplotlib.org/api/_as_gen/matplotlib.pyplot.boxplot.html

3.6 Dice coefficient loss

Milletari et al. (2016) introduced a loss function based on the dice score. Their loss function was also used for a medical image segmentation task. The dice coefficient loss function is given by:

$$-\frac{2 * P \cap L + s}{\sum P + \sum L} \tag{3.7}$$

Where P is the predicted voxels and L is the corresponding ground truth, s is a smoothing factor, it is set to 1 in this implementation. Note the negative sign since the goal during training is to minimize this function.

In this thesis dice score is used to score how the different architectures perform. Therefore, this loss function may be well suited for this task.

3.7 Set up

The U-Net and the CNN were trained and tested on one node on NTNU's EPIC supercomputer[7] using one Tesla P100 GPU from Nvidia with 16 GB of video memory. Jobs were submitted using the Slurm job scheduler[8]. EPIC made it possible to train and test many models at the same time. DeepMedic was trained and tested on the visual computing lab using Nvidia's Titan X GPU with 12 GB of video memory. Docker[9] had to be used to train and test on this computer.

The CNN and U-Net were implemented using Keras with a TensorFlow backend and the code can be found on GitHub[10]. Some of the functions used in this implementation have been copied from the original implementation of the CNN used in Kleesiek et al. (2016). These functions have been highlighted with comments. Kleesiek et al. (2016)'s solution was implemented with Theano[11] and it can be found on https://github.com/GUR9000/Deep_MRI_brain_extraction.

The images were produced using Nilearn[12]. The images with the brain mask and the raw MRI scan were produced with nilearn.plotting.plot_roi. The code for the statistics done in this master thesis can also be found on GitHub[13].

[7] https://www.hpc.ntnu.no/pages/viewpage.action?pageId=21037127
[8] https://slurm.schedmd.com/
[9] https://www.docker.com/
[10] https://github.com/oysteiae/BrainSegmentationStOlavs/
[11] http://deeplearning.net/software/theano/
[12] https://nilearn.github.io/
[13] https://github.com/oysteiae/StatisticsMaster

3.8 Experiments

3.8.1 Experiments with hyperparameters

These tests were done to see if some changes to the configurations of the CNN will make it perform better. Also, these experiments were done to test what configuration should be used for the U-Net. The configuration with the highest dice score was used for the rest of the experiments. All of these experiments were done with the OASIS data set with 60 % of the data used for training 20 % of the data used for validation and 20 % of the data used for testing. The evaluation of the configuration was done on the validation data and the testing data withheld for the other experiments.

Experiment with patch size

Increasing the CNN's patch size can lead to better results. With a bigger patch size, the network will have a greater context when predicting voxels. This experiment increased the patch size of the CNN to 119^3 for predictions and when training. Also, the CNN was tested with the original patch size of 59^3 for training and a patch size of 84^3 for prediction.

For the U-Net three different patch sizes was tried out: 32^3, 48^3 and 64^3.

Both architectures will use KLD as loss function.

Experiment with a different loss function

The dice score loss function introduced in Milletari et al. (2016) was tested on the CNN from Kleesiek et al. (2016) and the 3D U-net. The dice score loss function will be compared against the KLD loss function for both architectures. The U-Net used a patch size of (64^3) and the CNN used a patch size of (59^3).

3.8.2 Experiments with data sets and models

These experiments were done to test how each architecture behaves when they were trained and tested on data from different sources. The different deep learning architectures and the 3 different data sets presented earlier in this chapter were used for all of these experiments. For the first experiment DeepMedic extracted 5000 patches of size 17^3 per subepoch from the validation data, while in the rest of the experiments DeepMedic instead extracted 500 patches per subepoch for validation, this was done because of the increased time it would take to train DeepMedic when using 5000 patches instead of 500 patches. This has no effect on the results since the validation data was only used for logging during training. The CNN and U-Net extracts

1 sample from the validation set each update steps and calculates loss and accuracy on that sample.

The OASIS, LBPA40 and St. Olavs data sets were used for these experiments.

The accuracy graphs for training and validation data are presented in appendix A. Additionally, the feature maps for the first four layers of the different architectures are presented in appendix B.

Architectures trained and tested on one data set

Each architecture was trained on *one* data set and then tested on the same data set. 60 % of the data set was used for training, 20 % for validation and 20 % was used for testing. This was repeated so that every architecture is trained and tested on every data set.

Architectures trained on two data sets tested on the third

Each architecture was trained on *two* data sets and then tested on the remaining data set. The data set used for testing was also used to log validation data during training of the network. This experiment will highlight which architectures are better at generalizing to other data sets they have not seen before.

Architectures trained and tested on data from all three data sets

Each architecture was trained on all data from all three data sets. 60 % of the data set was used for training, 20 % for validation and 20 % was used for testing.

Architectures trained on equal amounts of data from each data set

To see if the ratio of the number cases in each data sets have an impact on the architectures' performance, this experiment trained and tested every architecture on 40 cases randomly chosen from each data set. The results will then be compared to the results in the previous experiment. 60 % of the data set was used for training, 20 % for validation and 20 % was used for testing.

Architectures trained on not resampled data

Each architecture was trained on all images in the different data sets. Data from LBPA40 and St. Olavs were not resampled for this experiment, while the OASIS data was unchanged since it was not resampled for the other experiments. In other words, the St. Olavs data set and the

LBPA40 data set will be used with their original voxel size. 60 % of the data set was used for training, 20 % for validation and 20 % was used for testing.

3.8.3 Testing CNN and the U-Net on different hardware configurations

The U-Net and the CNN were tested on different configuration to test how their training times vary with respect to the hardware configuration they were trained on. Two different batch sizes for the architectures were also tested to see what impact the increased load on the GPU will have on the training time. Only the OASIS data set was used for this experiment and no validation was used during training and no testing was done afterwards. Also, a patch size of 64^3 was used for the U-Net. This was done to further increase the load on the GPU. Table 3.6 details the tests that were done.

Table 3.6: *Different configurations for testing training time for U-Net and the CNN.*

Model	GPU	CPU	Batch size
CNN	1 Tesla P100	E5-2650 v4 12 cores 2.20GHz	4
CNN	2 Tesla P100	E5-2650 v4 12 cores 2.20GHz	4
CNN	1 Titan X	i7-6800K 6 cores 3.40GHz	4
CNN	1 GTX 1080	i7-6700 4 cores 3.40GHz	4
CNN	1 Tesla P100	E5-2650 v4 12 cores 2.20GHz	16
CNN	2 Tesla P100	E5-2650 v4 12 cores 2.20GHz	16
CNN	1 Titan X	i7-6800K 6 cores 3.40GHz	16
CNN	1 GTX 1080	i7-6700 4 cores 3.40GHz	16
U-Net	1 Tesla P100	E5-2650 v4 12 cores 2.20GHz	4
U-Net	2 Tesla P100	E5-2650 v4 12 cores 2.20GHz	4
U-Net	1 Titan X	i7-6800K 6 cores 3.40GHz	4
U-Net	1 GTX 1080	i7-6700 4 cores 3.40Hz	4
U-Net	1 Tesla P100	E5-2650 v4 12 cores 2.20GHz	8
U-Net	2 Tesla P100	E5-2650 v4 12 cores 2.20GHz	8
U-Net	1 Titan X	i7-6800K 6 cores 3.40GHz	8
U-Net	1 GTX 1080	i7-6700 4 cores 3.40GHz	8

Nvidia's Tesla P100[14], Nvidia's Titan X[15] and Nvidia's GTX 1080[16] used in this experiment are all based on Nvidia's Pascal architecture[17]. The Tesla P100 uses PCIe to connect to the motherboard instead of using NVlink. Using NVlink would have increased the clock speed of the GPU. Table 3.7 shows the specifications for the different GPUs.

[14] https://devblogs.nvidia.com/inside-pascal/
[15] https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/
[16] https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080/
[17] https://developer.nvidia.com/pascal

Table 3.7: *Comparison of the different GPUs.*

GPU	CUDA cores	Base clock	Boost clock	Memory	Memory bandwidth
P100	3584	1126 MHz	1303 MHz	16 GB	732 GB/s
Titan X	3584	1417 MHz	1531 MHz	12 GB	336.5 GB/s
GTX 1080	2560	1607 MHz	1733 MHz	8 GB	320 GB/s

The amount of CUDA cores in each GPU indicates how many computations the Nvidia GPU can do in parallel [18]. This is the case as long as the Nvidia GPUs are the same generation. Nvidia's GTX 1080 and the Titan X have *compute capability* 6.1 and Nvidia's Tesla P100 have *compute capability* 6.0. Compute capability is Nvidia's way of specifying their GPUs specifications and available features. This means that the amount of CUDA cores between the Titan X and the Tesla P100 cannot be directly compared while the amount of CUDA cores can be compared between the GTX 1080 and the Titan X.

There is much more science behind how different Nvidia GPUs process data and how they are designed. However, this is beyond the scope of this thesis.

The i7-6700 CPU have a clock speed of up to 4.00 GHz when demanding tasks are running, while the 6800K is capable of producing a clock speed of 3.60 GHz. However, on the computer which the 6800K is situated in lists the max clock speed as 3.8 GHz. It should also be noted that Intel's i7 6700 is not capable of overclocking while Intel's i7 6800K is. How much the 6800K is able to overclock differs from every unit produced. This should not be a big problem because the difference should be small, and the point of this experiment is to highlight the difference in training time for the different configurations not the absolute training time. Also, it is the GPU that has the most impact on the training times when training CNNs.

The configurations using the Titan X and the GTX 1080 GPU were trained on computers where more people can use them at the same time. Therefore, the training time for these can be influenced by other people using it while the model is training. However, this should not be a big problem since in this implementation TensorFlow, which is the computing backend used for the implementation of U-Net and the CNN, will allocate all of the GPUs memory[19] making it very hard to train other deep learning models at the same time.

3.8.4 Experiment on liver data

Only 11 CT images were used from the LiTS data set. This was done because this is only a feasibility study for investigating if the deep learning architectures used in this thesis are also

[18] https://developer.nvidia.com/cuda-faq
[19] https://www.tensorflow.org/programmers_guide/using_gpu

able to do liver segmentation. The data was resampled to voxel size $(1mm, 1mm, 1mm)$ and it was also normalized. The architectures were trained and tested on the same data.

Chapter 4

Results

This chapter presents the results from the experiments. First, the experiment where a different loss function and patch size was used for the CNN and the U-Net is presented. Next, the experiments with different data sets are presented. Then, the experiment showing the different training times when using different configurations is presented. Lastly, the experiment done with liver segmentation is presented.

4.1 Experiments with hyperparameters

4.1.1 Experiment with patch size

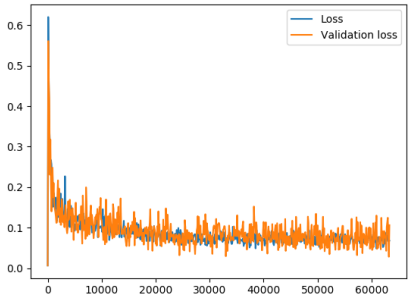
Table 4.1 shows the scores for the CNN and the U-Net when different patch sizes were tested. The table shows that the CNN with patch size 59^3 and the U-Net with patch size 32^3 had the highest dice score.

Table 4.1: *Dice score, sensitivity score and specificity score for CNN and U-Net different patch sizes.*

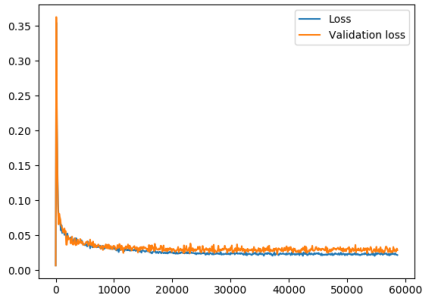
Architecture	Patch size	Dice score	Sensitivity	Specificity
CNN	(59, 59, 59)	0.94668 ± 0.017	0.96877 ± 0.013	0.97262 ± 0.013
CNN	(119, 119, 119)	0.90797 ± 0.021	0.97337 ± 0.010	0.94009 ± 0.016
U-Net	(32, 32, 32)	0.96376 ± 0.006	0.95824 ± 0.014	0.98941 ± 0.005
U-Net	(48, 48, 48)	0.95745 ± 0.005	0.94158 ± 0.011	0.99117 ± 0.005
U-Net	(64, 64, 64)	0.93782 ± 0.004	0.91888 ± 0.011	0.98578 ± 0.004

Figure 4.1 shows the loss and validation loss for the CNN with different patch sizes and figure 4.2 shows the loss and validation loss for the U-Net with different patch sizes. The x-axis

numbers the update steps during training. In figure 4.2 it can be seen that with a bigger patch size for the U-Net the training loss decreases while the validation loss increases.

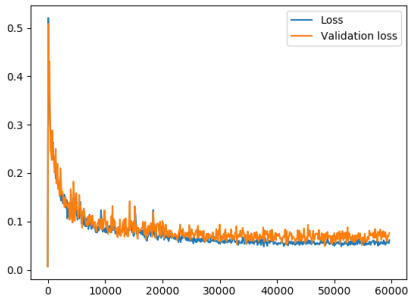


(a) *Loss and validation loss for the CNN with different patch size* (59^3).

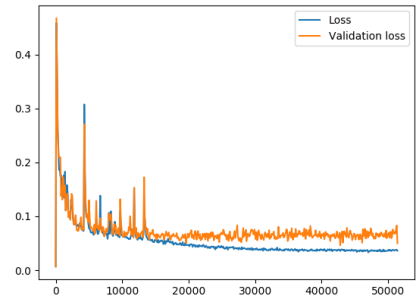


(b) *Loss and validation loss for the CNN with different patch size* (119^3).

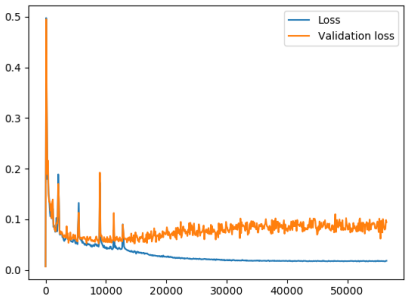
Figure 4.1: *Loss and validation loss for the CNN with different patch sizes.*



(a) *Loss and validation loss with patch size 32.*



(b) *Loss and validation loss with patch size 48.*



(c) *Loss and validation loss with patch size 64.*

Figure 4.2: *Loss and validation loss for U-Net with different patch sizes.*

4.1.2 Experiment with different a loss function

Table 4.2 shows the scores for the CNN and the U-Net when they used the dice loss function and when they used the KLD loss function. Both architectures have a higher score when the

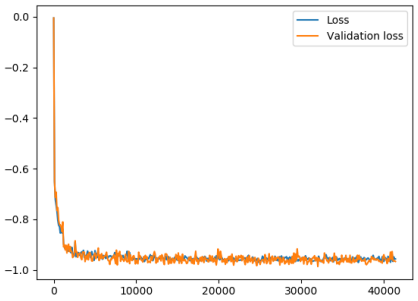
KLD loss function was used. The scores for the U-Net with the dice loss function shows that the loss function did not work for this implementation. The sensitivity score and the specificity score reveal that the U-Net predicts almost only brain voxels when using the dice loss function.

Table 4.2: *Scores U-Net different loss function.*

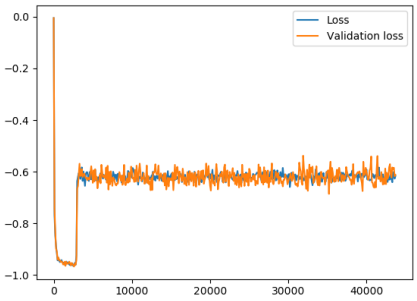
Dice score, sensitivity score and specificity score for the CNN and U-Net with different loss functions

Architecture	Loss function	Dice score	Sensitivity	Specificity
CNN	Dice loss	0.92302 ± 0.015	0.96729 ± 0.014	0.95494 ± 0.012
CNN	KLD	0.94668 ± 0.017	0.96877 ± 0.013	0.97262 ± 0.013
U-Net	Dice loss	0.41115 ± 0.000	1.00000 ± 0.000	0.00000 ± 0.000
U-Net	KLD	0.93782 ± 0.004	0.91888 ± 0.011	0.98578 ± 0.004

Figure 4.3 shows the loss and validation loss for the CNN and the U-Net. The x-axis numbers the update steps for the architectures. Figure 4.3b shows that the loss for the U-Net goes down until about the 1000th update step. After the 1000th update step the loss jumps up.



(a) *CNN with dice coefficient loss function.*



(b) *U-Net with dice coefficient loss function.*

Figure 4.3: *Dice coefficient loss on training data and validation data for the CNN and U-Net.*

The U-Net with the KLD loss function and patch size 32^3 had the highest dice score. Therefore, the U-Net with patch size 32^3 and the KLD as loss function will be used for the rest of the experiments. The CNN had the highest dice score when patch size 59^3 was used and KLD was used as loss function. The CNN will therefore use these hyperparameters for the rest of the experiments.

4.2 Experiments with data sets

For easier reading the highest score for each data set have been marked in bold for all of these experiments.

4.2.1 Architectures trained and tested on one data set

Tables 4.3, 4.4 and 4.5 show the dice, sensitivity and specificity scores for when the architectures were trained and tested on the same data set. The tables show that DeepMedic performed best in all metrics, except for the specificity score on the OASIS data set. The CNN had the highest score there.

Table 4.3: *Dice score for architectures trained and tested on the same data set.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	0.94542 ± 0.016	0.96346 ± 0.005	$\mathbf{0.96709 \pm 0.005}$
St. Olavs	0.95075 ± 0.014	0.88826 ± 0.026	$\mathbf{0.97432 \pm 0.006}$
LBPA40	0.95823 ± 0.011	0.98124 ± 0.001	$\mathbf{0.98503 \pm 0.001}$

Table 4.4: *Sensitivity for architectures trained and tested on the same data sets.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	0.96924 ± 0.015	0.96295 ± 0.012	$\mathbf{0.97386 \pm 0.010}$
St. Olavs	0.95956 ± 0.027	0.86211 ± 0.055	$\mathbf{0.97684 \pm 0.011}$
LBPA40	0.96778 ± 0.006	0.97804 ± 0.003	$\mathbf{0.98529 \pm 0.001}$

Table 4.5: *Specificity for architectures trained and tested on the same data set.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	0.97144 ± 0.014	$\mathbf{0.98745 \pm 0.004}$	0.98598 ± 0.005
St. Olavs	0.99351 ± 0.005	0.99155 ± 0.006	$\mathbf{0.99693 \pm 0.002}$
LBPA40	0.99014 ± 0.004	0.99708 ± 0.000	$\mathbf{0.99714 \pm 0.000}$

Figure 4.4 shows the box plot of the dice score for the architectures in this experiment. The box plot for each architecture shows the combined scores for when the architectures were trained and tested on the individual data sets. The U-Net box plot shows that it has the highest variance while the CNN and DeepMedic have less fluctuations in their scores.

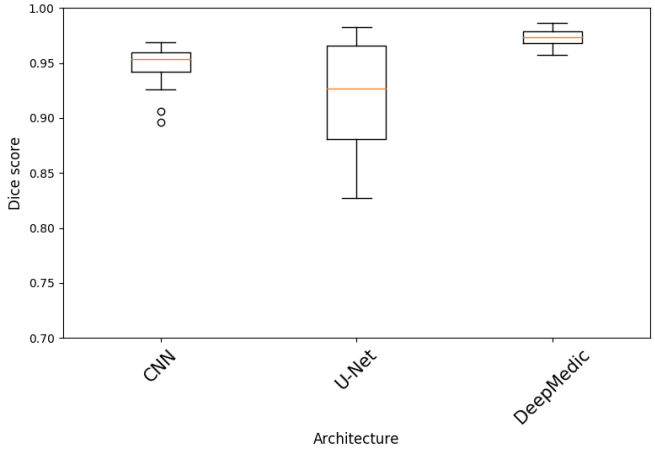


Figure 4.4: *Box plot of the dice scores for all the data when the architecture is trained and tested on the same data set.*

4.2.2 Architectures trained on two data sets and tested on the third

The tables here show the scores for the data that the architectures were not trained on. Table 4.6 shows that DeepMedic had the highest dice scores for all the data sets. Tables 4.7 and 4.8 show that the CNN and U-Net scored the highest in sensitivity and specificity on some data sets.

Table 4.6: *Dice score for architectures trained on two data sets and third.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	0.74791 ± 0.118	0.82923 ± 0.048	$\mathbf{0.93954 \pm 0.021}$
St. Olavs	0.75247 ± 0.131	0.77107 ± 0.117	$\mathbf{0.83920 \pm 0.091}$
LBPA40	0.81357 ± 0.039	0.81812 ± 0.028	$\mathbf{0.91370 \pm 0.022}$

Table 4.7: *Sensitivity for architectures trained on two data sets and tested on the third.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	0.61955 ± 0.144	0.75037 ± 0.071	$\mathbf{0.92864 \pm 0.045}$
St. Olavs	0.62907 ± 0.151	0.64820 ± 0.130	$\mathbf{0.74153 \pm 0.102}$
LBPA40	$\mathbf{0.99266 \pm 0.004}$	0.97709 ± 0.008	0.98613 ± 0.004

Table 4.8: *Specificity for architectures trained on two data sets and tested on the third.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	$\mathbf{0.99519 \pm 0.003}$	0.98055 ± 0.008	0.98356 ± 0.006
St. Olavs	0.99804 ± 0.004	$\mathbf{0.99844 \pm 0.003}$	0.99827 ± 0.002
LBPA40	0.91618 ± 0.022	0.92329 ± 0.015	$\mathbf{0.96783 \pm 0.009}$

Figure 4.5 show the box plot of the dice scores for when the architectures were trained on two data sets and tested on the third. Note that for this box plot the values for the y-axis are between 0 and 1. The box plot show that DeepMedic had less variance compared to the CNN and the U-Net. The CNN had most the variance according to the box plot.

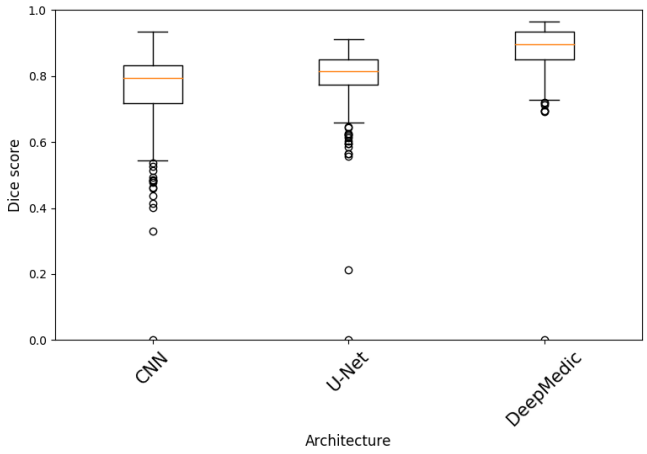


Figure 4.5: *Box plot of the dice scores for all the data when the architecture is trained on two data sets and tested on the remaining data set.*

4.2.3 Architectures trained and tested on data from all three data sets

The tables shown here show that DeepMedic had the highest dice score for all the data set and that it also had a high sensitivity and specificity score. Table 4.11 reveal that U-Net had the highest specificity score on two data sets.

Table 4.9: *Dice score for architectures trained and tested on all data from all three data sets.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	0.95754 ± 0.006	0.95999 ± 0.011	$\mathbf{0.96324 \pm 0.008}$
St. Olavs	0.94381 ± 0.0305	0.95715 ± 0.024	$\mathbf{0.96696 \pm 0.021}$
LBPA40	0.91892 ± 0.025	0.96447 ± 0.011	$\mathbf{0.97102 \pm 0.004}$

Table 4.10: *Sensitivity for architectures trained and tested on all data from all three data sets.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	$\mathbf{0.96252 \pm 0.018}$	0.95252 ± 0.027	0.96043 ± 0.022
St. Olavs	0.94635 ± 0.025	0.95597 ± 0.024	$\mathbf{0.97702 \pm 0.013}$
LBPA40	0.98034 ± 0.007	0.97472 ± 0.009	$\mathbf{0.98384 \pm 0.004}$

Table 4.11: *Specificity for architectures trained and tested on all data from all three data sets.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	0.98327 ± 0.008	$\mathbf{0.98897 \pm 0.005}$	0.98827 ± 0.006
St. Olavs	0.99385 ± 0.007	$\mathbf{0.99567 \pm 0.004}$	0.99556 ± 0.004
LBPA40	0.97215 ± 0.009	0.99166 ± 0.003	$\mathbf{0.99212 \pm 0.002}$

Figure 4.6 presents the box plot for when the architectures were trained and tested on all three data sets. In the box plot the U-Net and DeepMedic have less variance compared to the CNN.

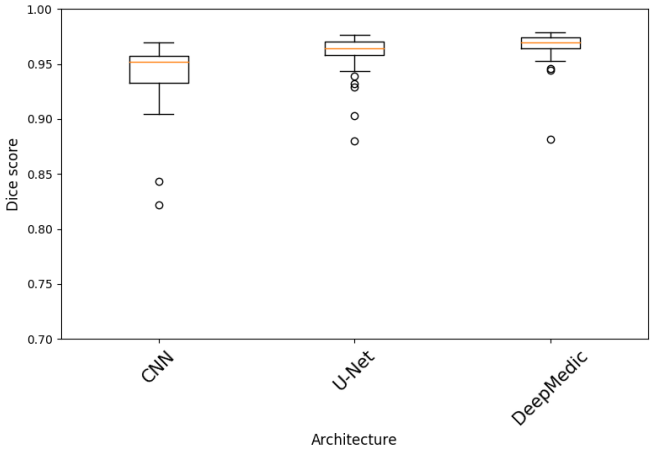


Figure 4.6: *Box plot of the dice scores for architectures trained and tested on all three data sets.*

Worst and best predictions for the architectures

The images show the best and worst predictions for the architectures when they were trained and tested on all the data. Green is correctly predicted brain voxels, red is falsely predicted brain voxels and cyan is falsely predicted non-brain voxels.

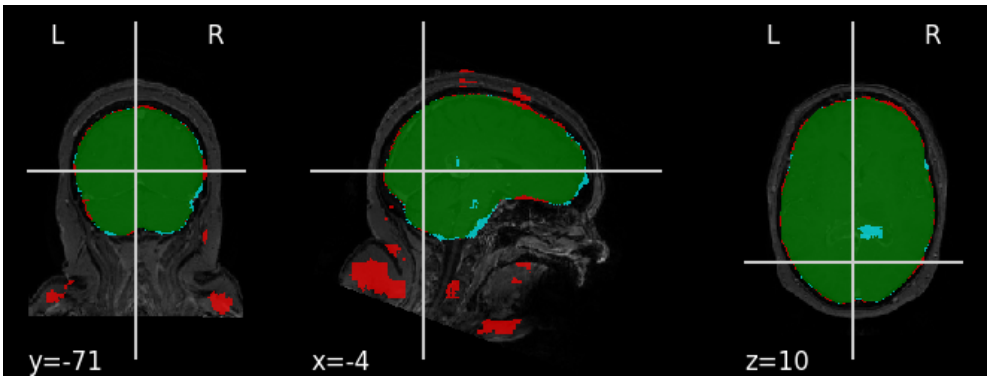


Figure 4.7: *237/coreg_T1.nii U-Net worst prediction.*

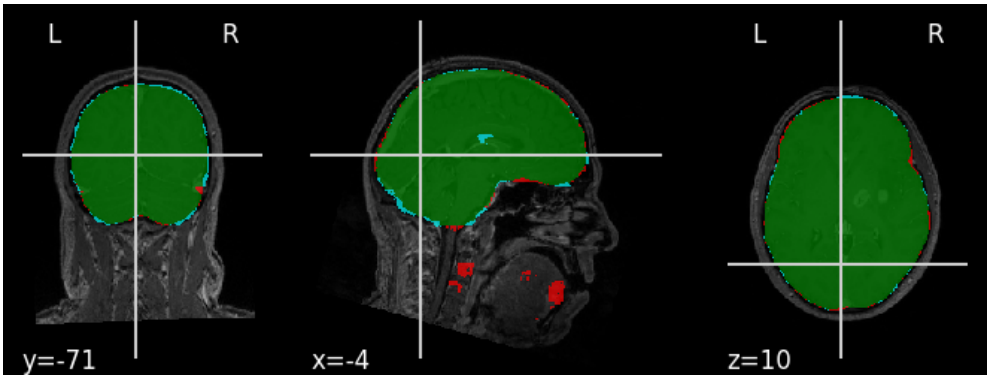


Figure 4.8: *97/coreg_T1.nii U-Net best prediction.*

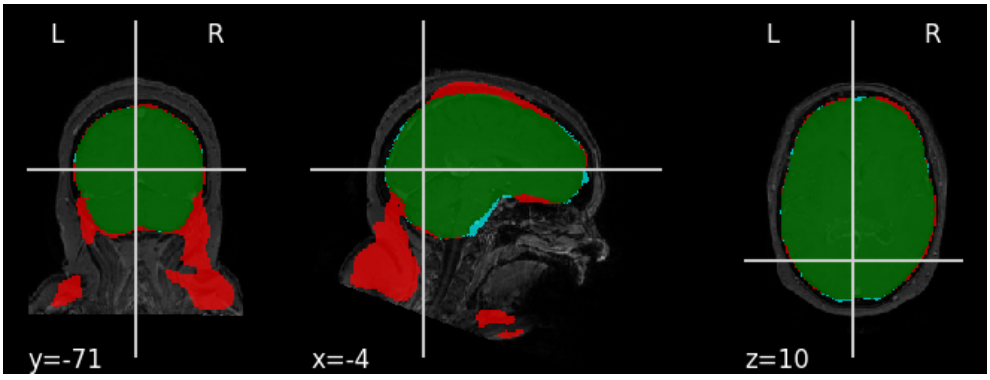


Figure 4.9: *237/coreg_T1.nii CNN worst prediction.*

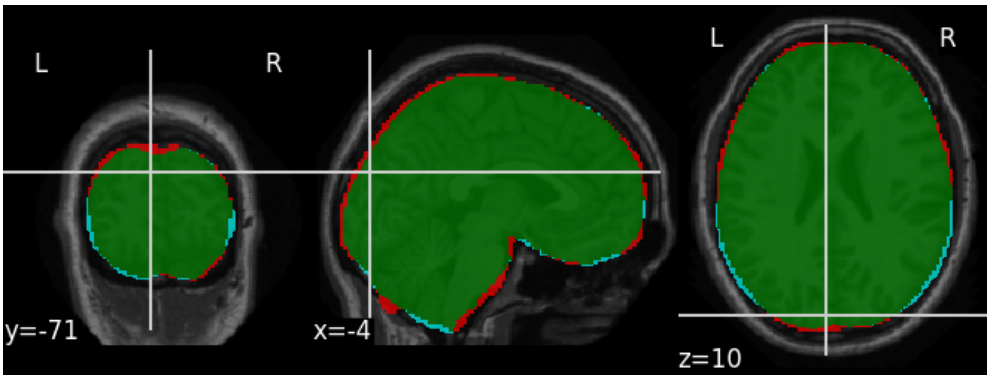


Figure 4.10: *OAS1_0012_MR1_mpr_n4_anon_111_t88_gfc CNN best prediction.*

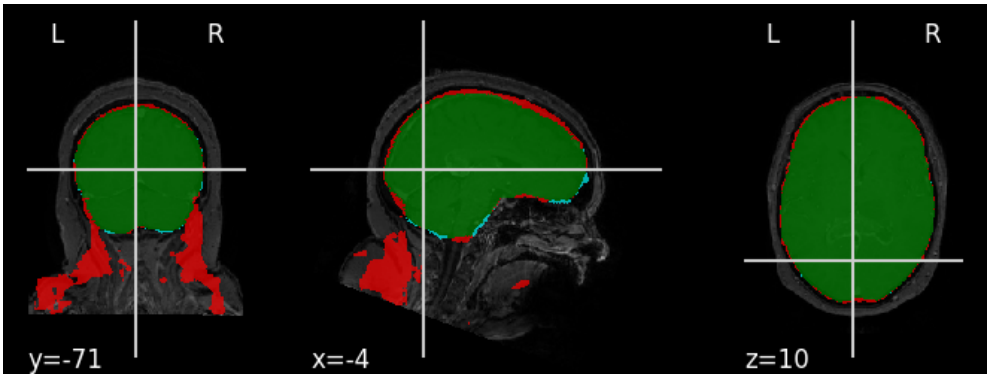


Figure 4.11: *237/coreg_T1.nii DeepMedic worst prediction.*

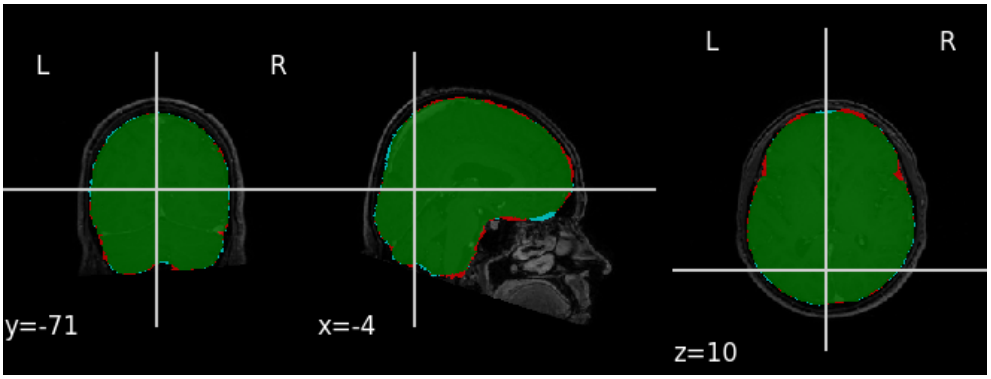


Figure 4.12: *17/coreg_T1.nii DeepMedic best prediction.*

Error maps

The figures in this section are the error maps for the architectures' predictions of the testing data from the experiment when all the data sets were used for training. Note that the original images are positioned differently in 3D space causing for example the amount wrongly predicted voxels at the edges of the brain to seem off. The colors in the image detail the amount of wrongly predicted voxels, the scale on the right of the image specifies what number the different colors correspond to.

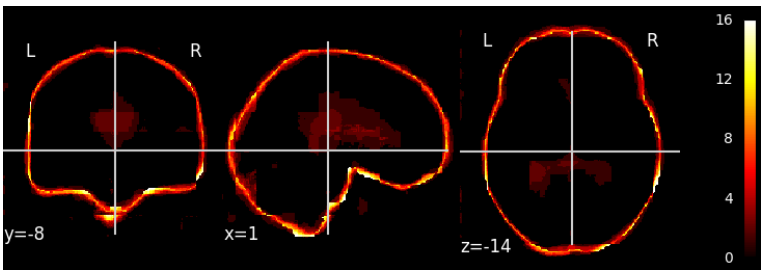


Figure 4.13: *Error map for U-Net predicting the OASIS data.*

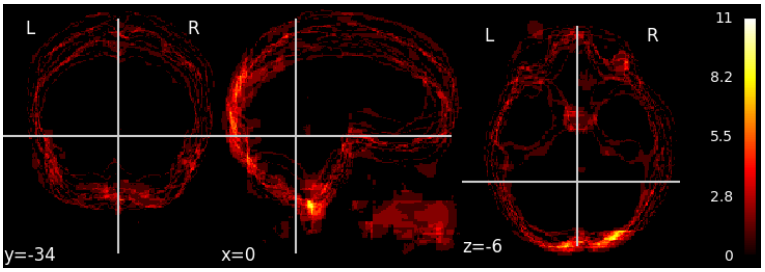


Figure 4.14: *Error map for U-Net predicting the LBPA40 data.*

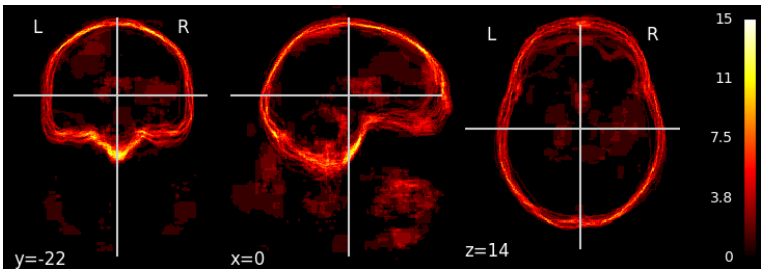


Figure 4.15: *Error map for U-Net predicting the St. Olavs data.*

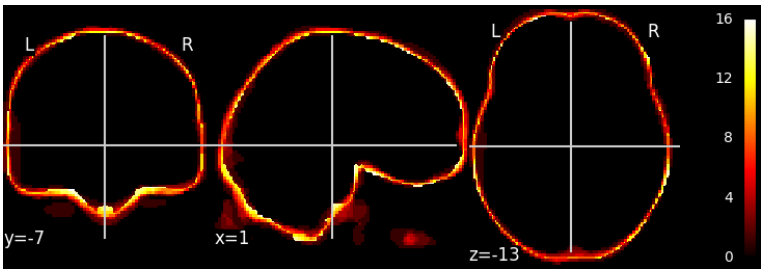


Figure 4.16: *Error map for the CNN predicting the OASIS data.*

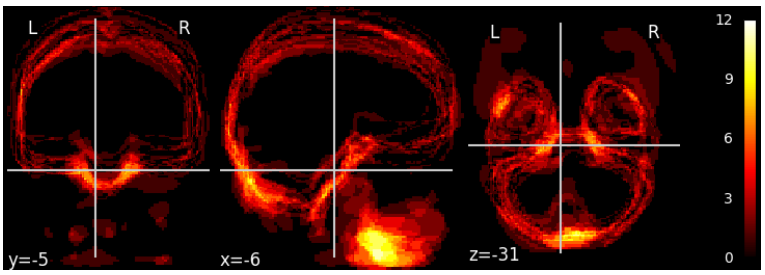


Figure 4.17: *Error map for the CNN predicting the LBPA40 data.*

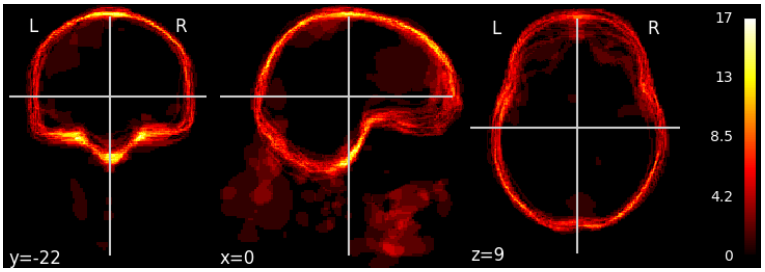


Figure 4.18: *Error map for the CNN predicting the St. Olavs data.*

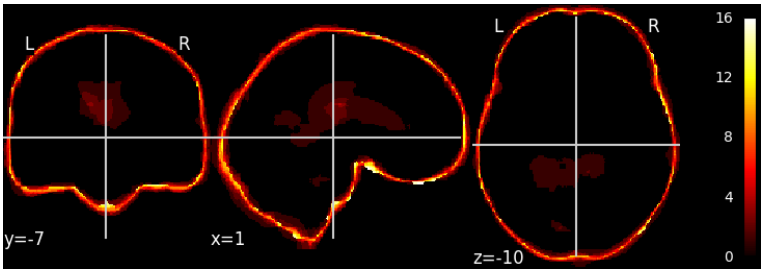


Figure 4.19: *Error map for DeepMedic predicting the OASIS data.*

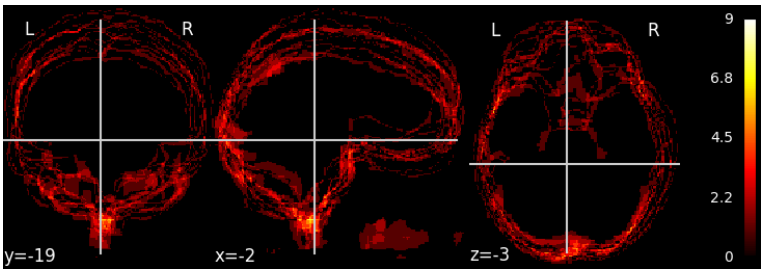


Figure 4.20: *Error map for DeepMedic predicting the LBPA40 data.*

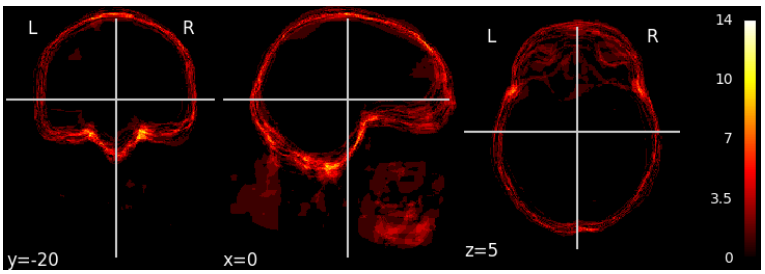


Figure 4.21: *Error map for DeepMedic predicting the St. Olavs data.*

4.2.4 Architectures trained on equal amounts of data from each data set

In tables 4.12, 4.13 and 4.14 DeepMedic has the highest scores on almost every data set. The architectures scored lower on the St. Olavs data in this experiment compared to the last experiment when the architectures were trained on all data from all three data sets.

Table 4.12: *Dice score for architectures trained and tested on equal amounts of data from each data set.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	0.95531 ± 0.007	0.92728 ± 0.012	$\mathbf{0.96510 \pm 0.003}$
St. Olavs	0.85565 ± 0.136	0.73717 ± 0.105	$\mathbf{0.94761 \pm 0.037}$
LBPA40	0.95237 ± 0.007	0.93699 ± 0.008	$\mathbf{0.97963 \pm 0.003}$

Table 4.13: *Sensitivity for architectures trained and tested on equal amounts of data from each data set.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	$\mathbf{0.97176 \pm 0.010}$	0.92074 ± 0.037	0.96937 ± 0.012
St. Olavs	0.78969 ± 0.201	0.62618 ± 0.149	$\mathbf{0.92737 \pm 0.076}$
LBPA40	0.97242 ± 0.006	0.95930 ± 0.018	$\mathbf{0.98326 \pm 0.006}$

Table 4.14: *Specificity for architectures trained and tested on equal amounts of data from each data set.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	0.97806 ± 0.008	0.97748 ± 0.008	$\mathbf{0.98622 \pm 0.004}$
St. Olavs	$\mathbf{0.99760 \pm 0.002}$	0.99456 ± 0.005	0.99705 ± 0.001
LBPA40	0.98652 ± 0.003	0.98292 ± 0.004	$\mathbf{0.99530 \pm 0.002}$

Figure 4.22 shows the box plot for when the architecture were trained on equal amounts of data from each data set. Note that for this box plot the values for the y-axis are between 0.5 and 1. The box plot shows that the U-Net had the highest variance for its scores in this experiment.

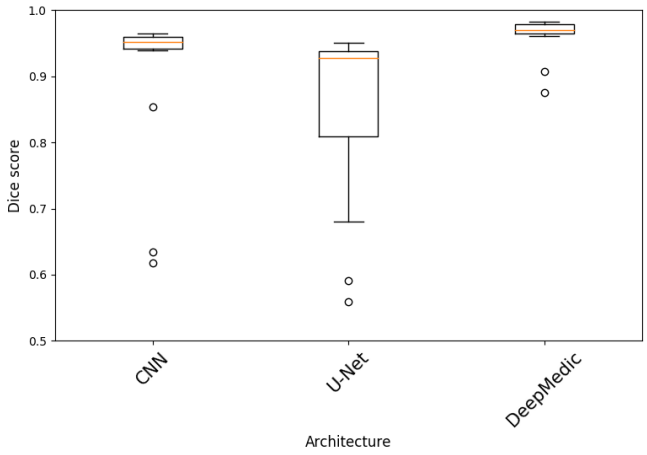


Figure 4.22: *Box plot of the dice scores when the architecture are trained on equal amounts of data from each data set.*

4.2.5 Architectures trained on not resampled data

Table 4.15 shows that DeepMedic had the highest dice score for all the data sets when the St. Olavs data set and the LBPA40 data set were not resampled. The architectures had lower dice scores for the St. Olavs data set in this experiment compared to the experiment where every architecture was trained and tested on all three data set, and the St. Olavs data set and the LBPA40 data set were resampled.

Table 4.15: *Dice score for architectures trained and tested on not resampled data.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	0.95829 ± 0.007	0.95344 ± 0.010	$\mathbf{0.96201 \pm 0.006}$
St. Olavs	0.93686 ± 0.033	0.92975 ± 0.036	$\mathbf{0.95446 \pm 0.021}$
LBPA40	0.92615 ± 0.008	0.95587 ± 0.014	$\mathbf{0.97210 \pm 0.003}$

Table 4.16: *Sensitivity for architectures trained and tested on not resampled data.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	0.96452 ± 0.018	0.95737 ± 0.023	$\mathbf{0.96934 \pm 0.016}$
St. Olavs	0.92709 ± 0.059	0.91307 ± 0.058	$\mathbf{0.96834 \pm 0.031}$
LBPA40	0.94091 ± 0.027	0.96692 ± 0.016	$\mathbf{0.97886 \pm 0.008}$

Table 4.17: *Specificity for architectures trained and tested on not resampled data.*

Data set	CNN	3D U-Net	DeepMedic
OASIS	0.98302 ± 0.009	0.98222 ± 0.010	$\mathbf{0.98397 \pm 0.006}$
St. Olavs	0.99142 ± 0.006	$\mathbf{0.99165 \pm 0.005}$	0.98974 ± 0.005
LBPA40	0.98471 ± 0.004	0.99058 ± 0.004	$\mathbf{0.99382 \pm 0.002}$

Figure 4.23 shows the box plot for when the architectures were trained and tested on all the data sets and the St. Olavs data set and the LBPA40 data were not resampled. Every architecture had more variance in this box plot compared to the box plot in figure 4.6 which shows the box plot for when the architectures were trained and tested on all three data sets and the data from St. Olavs and the data from LBPA40 were resampled.

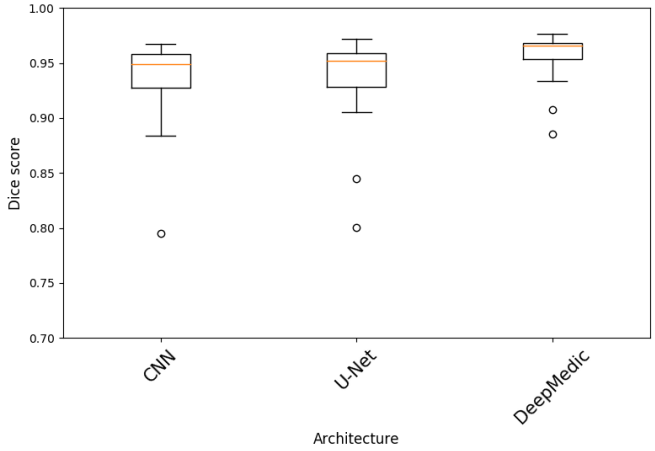


Figure 4.23: *Box plot of the dice scores when the architecture are trained on data that was not resampled.*

4.3 Testing CNN and the U-Net on different hardware configurations

4.3.1 CNN

Figure 4.24 and table 4.18 show that when training the CNN with batch size 4 the configuration using the Titan X was the fastest followed by the configuration using the GTX 1080.

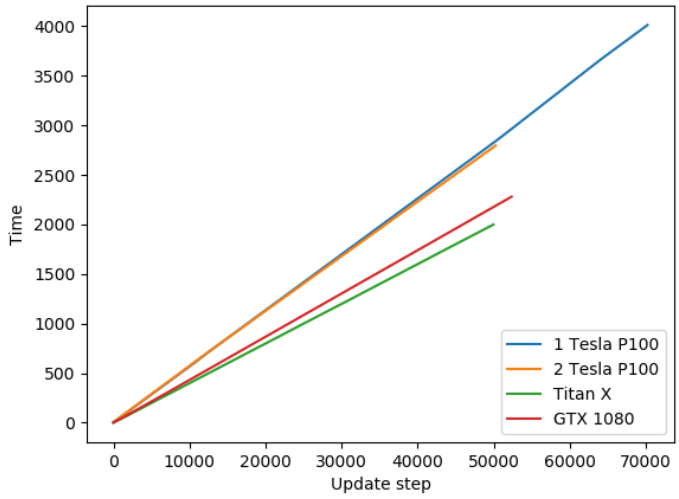


Figure 4.24: *The CNN with batch size 4.*

Table 4.18: *Average time for each update on the different configuration for the CNN with batch size 4.*

Configuration	Average time in seconds per update
1 Tesla P100	0.05708
2 Tesla P100	0.05554
Titan X	0.03999
GTX 1080	0.04354

The following graph and table show that the Titan X was the fastest configuration when training the CNN with batch size 16. The configuration using 2 Tesla P100 was the second fastest.

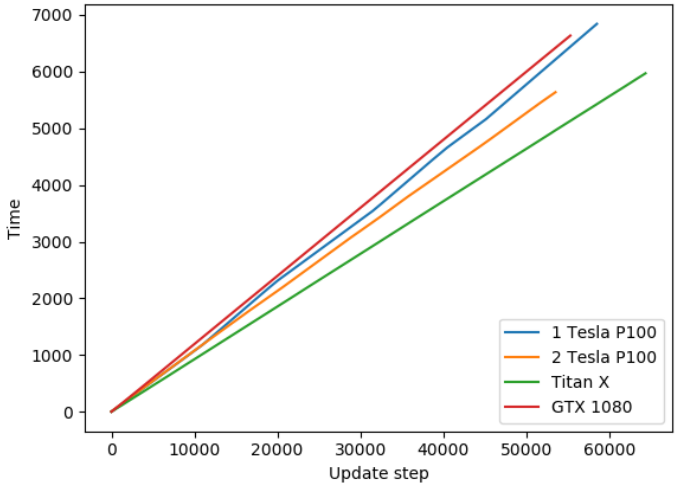


Figure 4.25: *The CNN with batch size 16.*

Table 4.19: *Average time for each update on the different configuration for the CNN with batch size 16.*

Configuration	Average time in seconds per update
1 Tesla P100	0.11689
2 Tesla P100	0.10520
Titan X	0.09270
GTX 1080	0.11991

4.3.2 U-Net

This figure and table presents the training time for when the U-Net was trained on different configuration using batch size 4. They show that using the configuration using the Titan X was

the fastest configuration.

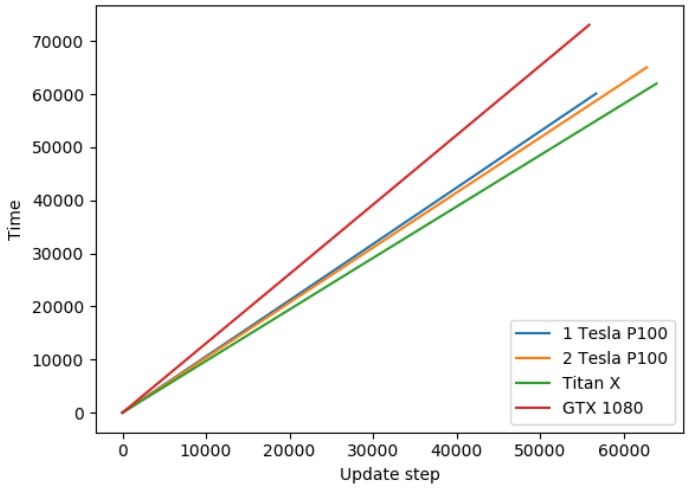


Figure 4.26: *3D U-net with batch size 4.*

Table 4.20: *Average time for each update on the different configuration for the U-Net with batch size 4.*

Configuration	Average time in seconds per update
1 Tesla P100	1.05999
2 Tesla P100	1.03612
Titan X	0.96948
GTX 1080	1.30704

The GTX 1080 could not run the U-Net with batch size 8 because it does not have a sufficient amount of memory. The result for this configuration is therefore not presented here. Figure 4.27 and table 4.21 show that the configuration using 2 Tesla P100 was the fastest configuration. This was the only test where the Titan X was not the fastest configuration.

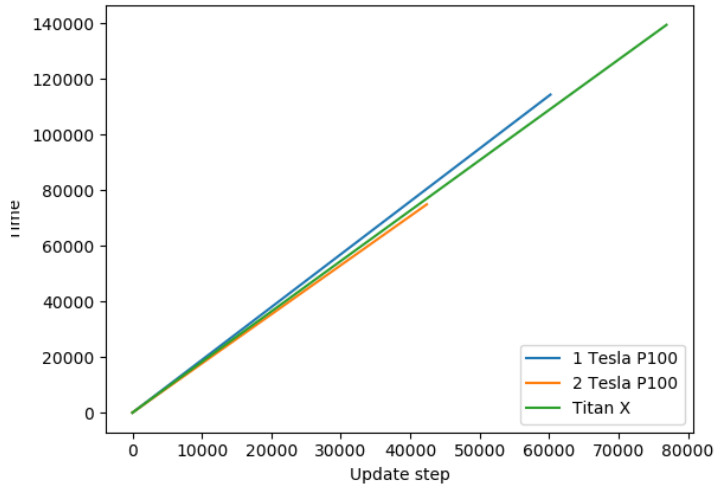


Figure 4.27: *3D U-net with batch size 8.*

Table 4.21: *Average time for each update on the different configuration for the U-Net with batch size 8.*

Configuration	Average time in seconds per update
1 Tesla P100	1.89850
2 Tesla P100	1.76553
Titan X	1.81191

4.4 Experiment on liver segmentation

Table 4.22: *Dice, sensitivity and specificity scores on the LiTS data set.*

Metric	CNN	3D U-Net	DeepMedic
Dice score	0.60768 ± 0.123	0.00007 ± 0.000	0.81087 ± 0.051
Sensitivity	0.60470 ± 0.155	0.00003 ± 0.000	0.98697 ± 0.004
Specificity	0.99234 ± 0.002	1.00000 ± 0.000	0.98958 ± 0.004

The following figures show the best predictions for the CNN and DeepMedic on the LITS data. The U-Net prediction is omitted here since it predicted only non-liver voxels for all the CT images. Green is correctly predicted liver voxels, red is falsely predicted liver voxels and cyan is falsely predicted non-liver voxels.

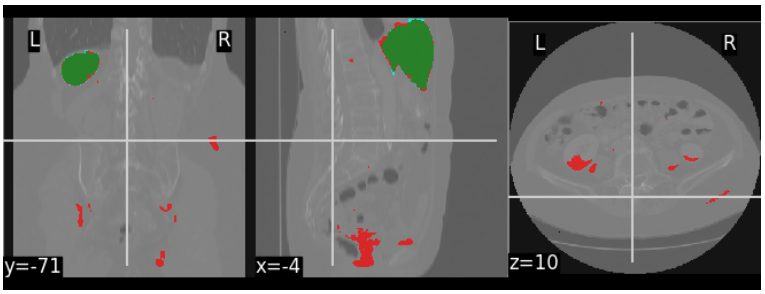


Figure 4.28: *Best prediction of LiTS data set from DeepMedic.*

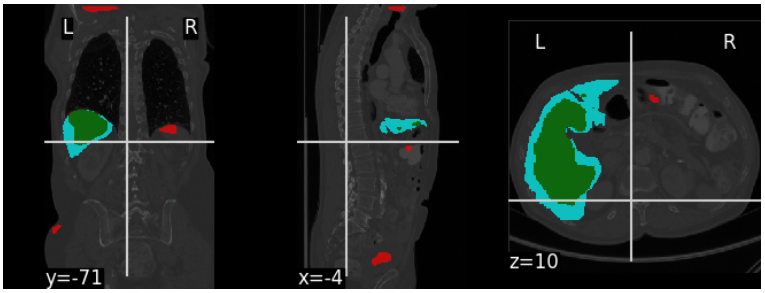


Figure 4.29: *Best prediction on LiTS data set from CNN.*

Chapter 5

Discussion

This chapter discusses the results of the different experiments. The different deep learning architectures will be compared, and their good and bad sides will be discussed. The first section will discuss the parameter experiments for the U-Net and the CNN. The second section will discuss how the different deep learning architectures performed skull stripping on the data sets. The next section will look at how the different architectures performed on different hardware configurations. The fourth section will discuss how the architectures performed when doing liver segmentation. The final section is the reflection.

5.1 Experiments with hyperparameters

5.1.1 Experiment with patch size

It seemed that increasing the patch size for the CNN did not improve its scores much. The loss graphs in figure 4.1 suggests that the bigger patch size made the network overfit the data because the validation loss seems to be a bit higher compared to the training loss. The loss also seems to be much more stable with the bigger patch size. The CNN with the bigger patch had a higher sensitivity compared to the CNN with the smaller patch size, but a much lower specificity. This indicated that the larger patch size made the CNN produce more false positive. With a smaller patch size the CNN is more accurate in its predictions since it predicts fewer voxels in each forward pass. Apparently, the lower patch size is large enough so that the architecture can get enough context for its predictions when doing skull stripping.

The U-Net benefited from having a smaller patch. The sensitivity and specificity scores in table 4.1 show that the U-Net with a smaller patch size got a higher sensitivity score compared to U-Net with bigger patches, but the specificity scores seemed to be close. This indicates that the smaller patch size made U-Net better at predicting true positive brain voxels. The loss graphs in

figure 4.2 shows that U-Net with a patch size of 64^3 is clearly overtraining on the data since it has an increasingly better loss for the training data, but the loss for the validation data becomes increasingly worse. The overtraining is also apparent in the loss graph for U-Net with patch size 48^3.

The reason why a bigger patch size makes the U-Net overtrain could be because the network will see a higher portion of the data earlier in the training session. With a smaller patch size it takes a longer time for the network to see patches of the data multiple times making it harder to overtrain. The training of the U-Net was the same as the training scheme used in Kleesiek et al. (2016). Maybe this was not the best choice and instead other ways of training should be explored for the U-Net. There were also no regularization methods implemented except for artificially expanding the training data by adding and multiplying the input with random values. If some simple regularization methods such as L1 or L2 regularization were added, this implementation of U-Net would perhaps not overtrain as much with bigger patch sizes.

5.1.2 Experiment with a different loss function

The CNN did not get a better dice score with the dice coefficient loss function. There could be many reasons for this. The first one is that the training implementation was not suited for the use of the dice coefficient loss function or that KLD is a loss function that is better suited for doing skull stripping.

The dice coefficient loss function did not work for this implementation of U-Net. The scores in table 4.2 shows that the U-Net only predicts brain-voxels. Figure 4.3b shows that the U-Net is not learning at all. At approximately the 1000th update step the loss jumps up and it stays there for the rest of the training session. One reason for this could be that the learning rate was too high.

5.2 Experiments with data sets and models

5.2.1 Architectures trained and tested on one data set

For the first set of experiments where each model was trained and tested on the same data set the DeepMedic architecture got the highest dice score on all experiments as table 4.3 shows. The U-Net architecture was relatively close on the OASIS data set and the LBPA40 data set. On the St. Olavs data set the U-Net got a dice score of 0.88 which was much lower compared to the other architectures scores on the data set. This could be due to the U-Net overfitting on the training data set. The accuracy graph in figure A.5 shows some indication of this since the accuracy on the validation data seems to be a tad lower compared to the training data. The CNN

had the lowest score on the OASIS data set and the LBPA40 data, but it had a higher score on the St. Olavs data set compared to U-Net by a fair margin.

The DeepMedic architecture also had the best sensitivity score on all the different data sets as can be seen in table 4.4. The CNN had better sensitivity scores compared to the U-Net except on the LBPA40 data set. The sensitivity scores for U-Net also gives an indication of why it had a low dice score on the St. Olavs data. It seems that it had problems with predicting true positive brain voxels on this data set. This seems to be a problem on most of the data sets for U-Net since it had on average the worst sensitivity scores.

The specificity scores in table 4.5 suggests that all the architectures are very good at predicting true negatives. Every architectures had a score of 0.99 + except for on the OASIS data set where every architecture had a somewhat lower score with the CNN having the lowest score. Overall, the DeepMedic again had the best specificity scores other than on the OASIS data set where the U-Net had the highest score. Since the U-Net also had a high specificity score on the St. Olavs data set, this confirms that the reason for the low dice score on the St. Olavs data set is because of its problems with predicting true positive brain voxels.

The dice score box plot in figure 4.4 shows that DeepMedic not only had the highest score, but its scores also fluctuated less compared to the other architectures' score. It had no outliers and all of its scores were between 0.95 and 0.99. The CNN also had good consistency while the dice scores for the U-Net were very inconsistent because of its performance on the St. Olavs data set.

5.2.2 Architectures trained on two data sets tested on the third

In the experiment were the architectures were trained on two data sets and tested on the remaining data set, the DeepMedic architecture again seemed to be the best architecture. Table 4.6 shows that DeepMedic achieved a dice score higher than the other architectures on all data sets. The CNN had the worst dice scores overall. DeepMedic struggled most on the St. Olavs data set. The reason for this is most likely because the St. Olavs data is the the data set with the most variability and also the data set that is, probably, the most different from the other two data sets considering that all of the patients have tumors.

Table 4.7 shows that, interestingly, all the architectures had a high sensitivity score for the LBPA40 data set compared to the other data sets, with the CNN scoring the highest. As they still had a low dice score on the LBPA40 data set, this shows that the architectures predicted many false positive brain voxels. The reason for this is most likely because the MRI scan in the LBPA40 data set contain image slices all the way down to the neck, in contrast to the OASIS scans and many of the scans in the St. Olavs data set where the slices go down to the start of the neck. The architectures were not trained to cope with this extra region of the MRI scan causing them to predict brain voxels where there should not be any.

Generally, the architectures were good at predicting true negative voxels for the OASIS and the St. Olavs data set. The low dice scores on these data sets can then mostly be attributed to the architectures not being good at predicting true positive brain voxels for these data sets.

The results for DeepMedic in this experiment further highlights that it is the best architecture used in this thesis. It had a higher dice score by a fair margin compared to the other architectures. Furthermore, the disparity for the dice scores increased a lot compared to the previous experiment. The reason for this can be that the DeepMedic uses L1 and L2 regularization as well as dropout. This should hinder the DeepMedic from overfitting the data it is training on. Lastly, it could be because of the the larger context of the image that the DeepMedic can infer for each patch it predicts.

Even though the architectures were not trained on the data set, they performed better than chance. This shows that some skull stripping insight is transferable when doing predictions on other data sets. However, the markedly lower scores compared to the previous experiment shows the importance of having the networks trained on the data sets that they are supposed to do skull stripping on. Further, because of the low training scores on the St. Olavs data set, training on data sets with tumors also seem to be very important for the different architectures.

The answer to **RQ 1** is then that it is important for architectures to train on data from the same source, or at least similar data, before performing skull stripping. This was the case for every architecture. For DeepMedic this was less important, but its dice scores were still not satisfactory. Not surprisingly, regularization may be the reason for why DeepMedic was better at predicting data from an unseen source. Regularization could have helped the architecture overfit its training data less compared to the other architectures.

5.2.3 Architectures trained and tested on data from all three data sets

The experiment where the architectures were trained and tested on all the data again showed that the DeepMedic was the best architecture at predicting brain masks. Its scores got somewhat worse compared to when it was trained and tested on each data set individually, but it is not very notable. The dice scores for the CNN got worse, except on the OASIS data set, as table 4.3 and table 4.9 shows. The CNN got a much worse result on the LBPA40 data set with the dice score decreasing 0.39 on average. Since the CNN got a higher sensitivity and a lower specificity, this can be attributed to the CNN being worse at predicting non-brain voxels. On the OASIS data set, however, it seems that the CNN got a higher dice score because of the increase in specificity. Training the CNN on more data sets made it better at predicting non-brain voxels on the OASIS data set while it was the opposite for the LBPA40 data set.

The U-Net got a much better score on the St. Olavs data set when it was trained on all the data sets compared to when it was just trained on just the St. Olavs data set. Having more data to train on could have caused the architecture not to overtrain on the data set as much, which again

led to the higher scores.

The Box plot 4.6 shows again that DeepMedic had the most stable scores. The architecture had only two outliers. The dice scores for the U-Net architecture were also much more stable compared to its scores in the first box plot in figure 4.4. This can be attributed to it getting higher scores on the St. Olavs data set. The CNN had the worst scores with the lowest average and median. It also had the most inconsistency in its dice scores.

The figures 4.7, 4.9 and 4.11, which shows the worst predictions for each architecture, show that every architecture struggled with MRI scan number 237 from the St. Olavs data set. One explanation for this is that on the image there seems to be a growth on the patient's neck. In figure 4.8, which shows the best prediction for U-Net, the patient does not seem to have this growth on his or her neck. The architecture struggles with the growth because they have not seen this type of image during training, making them unprepared when faced with the image during testing. This demonstrates that it is very important for the architectures to have training sets which contains "unusual" examples if they are to do good predictions on data that is atypical. Apart from the spurious predictions in the neck region of the patient the brain mask seems to be mostly fine.

One problem that is apparent for the U-Net in figures 4.7 and 4.8 is that it has falsely predicted non brain voxels in some center regions of the brain. This does not seem to be a problem for the CNN or the DeepMedic. One reason for this could be that the U-Net does predictions on all the voxels in the input. In contrast, the CNN and DeepMedic predict the central voxels for the input. This leads to the DeepMedic and the CNN having more context for each predicted voxel causing them to have less of a problem with false negative holes in the predicted brain mask. Of course, this implementation of U-Net also predicts on overlapping patches of the image to rectify this, but it seems that this does not work every time.

The error maps in section 4.2.3 shows which region of the MRI scans that the architectures struggle the most on when they are trained and tested on all three data sets. The figures show how the data sets are different and how the architectures encounter different problems in the different data sets. The images demonstrate that, for all the architectures, the borders of the brain mask is the most difficult to predict. This is no surprise since this should be the hardest region considering that there are more complex lines and structures there.

For the OASIS data, the U-Net and DeepMedic seem to have almost only wrongly predicted voxels on the border of the brain. The CNN, however, seems to also have some false positives in the lower right-hand corner of the sagittal slice. On the LBPA40 data, every architecture had problems with wrongly predicted brain voxels for this region, with the CNN seemingly struggling the most with this region. For the St. Olavs data, every architecture seemed to have problems with regions in the neck and the mouth.

5.2.4 Architectures trained on equal amounts of data from each data set

Training and testing on equal amounts of data from each data set made the U-Net worse at predicting on all data sets as table 4.12 shows. The CNN got worse results on the St. Olavs data set, but it got similar scores for the OASIS data set and much better scores on the LBPA40 data set. DeepMedic had similar results on the OASIS data set, got a bit higher score on the LBPA40 data set and it also got worse results on the St. Olavs data. One reason that explains why every architecture got a worse score on the St. Olavs data set is again that this data set is more variable compared to the other data sets. Also, the architectures were trained less on the tumors in the MRI scans from St. Olavs. When less of the St. Olavs data set was used for training, the architectures were less equipped for predicting uncommon structures in the MRI scans.

The U-Net scores are, as mentioned, worse across the board compared to the experiment when it was trained on all the data. Table 4.13 shows that the reason for this is because it was bad at predicting true positive brain voxels. This could be because of the reason mentioned earlier when discussing why the U-Net had random falsely predicted negative voxels in its predicted brain masks. This effect could have been magnified when predicting on less data from each data set.

RQ 2 asked if the balance between the amount of data in each data set was important for deep learning architectures when doing skull stripping. The answer to this is that there could be some improvements for the smallest data set if the amount of data in the other data set was the same. This is because both DeepMedic and the CNN got a higher score on the LBPA40 data set, which had the lowest number of images in it in the previous experiment. However, since the U-Net got a worse score on this data set this might not always be the case. As for the for the largest data set, the data from St. Olavs, the scores got worse indicating that for large data sets with brain tumors or other factors that causes high variability in the scans, it is best to train on all the available data. The highest scores for each data set was mostly obtained when training and testing on only one data set, indicating that the architectures should train on only one data set for best performance.

5.2.5 Architectures trained on not resampled data

The experiment where the networks where trained and tested on data that was not resampled showed a decline in performance for the U-Net on every data set except for the OASIS data set, as can be seen in table 4.15. The DeepMedic architecture only seemed to be worse at predicting on data from the St. Olavs data set, while the CNN had a slight improvement on the LBPA40 data set and the OASIS data set. It is expected that the architectures would be worse at predicting data from the St. Olavs data set because when the data is not resampled, the data is even more variable.

The answer to **RQ 3** is then that for data sets that are very variable it is important to resample the MRI scans to the same voxel size for optimal performance of the deep learning architectures. The scores for the LBPA40 and the OASIS data set indicates that for less variable data sets resampling is not as important, but for the scores for the St. Olavs data set were worse showing that resampling is an important preprocessing step when doing skull stripping.

5.2.6 General

These experiments have shown that it is important for the models to be trained on data sets it is supposed to predict on. Even though all the data sets were resampled to voxel size $(1mm, 1mm, 1mm)$. One of the reasons for this can be seen in figure 3.4 and 3.2. The LBPA40 data set contains images where more of the patient's neck is part of the image. When the models have not seen these parts of the MRI scan it can cause them to produce bad predictions. This is especially the case for the patch based architectures used in this thesis. Since they cannot infer context from other parts of the image if the input is only neck voxels. This error is also seen in the architectures' error maps for the LBPA40 data. All the architectures struggle somewhat with regions in the neck, especially the CNN.

It should also be noted that the training configuration for the DeepMedic is different from the training configuration of the other two architectures. DeepMedic includes regularization techniques such as L1 and L2 regularization and dropout so that the model is less prone to overfitting the data. This is most likely the reason for why DeepMedic had the most stable scores. U-Net on the other hand was erratic indicating that more regularization techniques should be added to this implementation. It had high scores when it was tested on the St. Olavs data when it was trained on all data set, but when it was only trained on the St. Olavs data set it had a very low score.

These results coincide with the results that Casamitjana et al. (2016) got in their work where they compared a fully convolutional network, a U-Net architecture and a DeepMedic like architecture on brain tumor data. Although, it should be noted that their architectures were designed to do full predictions in one forward pass. Their DeepMedic like architecture also got the best results in their experiments. However, the experiment done in this master thesis was not perfect because DeepMedic, which is the multi scale architecture, includes more sophisticated elements such as different regularization methods which both the CNN and the U-Net lacks.

Furthermore, the U-Net was originally designed to do full predictions in one forward pass. Because of hardware limitations and time limitations this was not done for this implementation of U-Net.

The dice scores in tables 4.9 and 3.5 indicates that DeepMedic might be a better method for performing skull stripping compared to the implementation in Kleesiek et al. (2016). The experiments done in this thesis and Kleesiek et al. (2016) are, however, different and also these

experiments used different data sets, which means that this is not certain and more testing have to be done to conclude on anything.

The CNN is by far the fastest for training and predicting. Training can take only $1-2$ hours on a Nvidia TitanX, no matter how large the data set is and predicting one brain mask takes a few seconds depending on how large the image is. Training DeepMedic takes much longer time. For example, when DeepMedic was used to train on all the data sets, it took 50 hours to train on a Nvidia TitanX. However, predicting one brain mask for DeepMedic takes only under one minute. The U-Net takes approximately $3-4$ hours to train on a Nvidia TitanX. The prediction takes much longer compared to the other architectures because it uses the overlapping technique to do predictions. The prediction time really depends on the data, but it takes approximately 2 minutes to predict one brain mask.

Based on these experiment DeepMedic seems to be the best skull stripping solution for the St. Olavs hospital. Again, this can be due the U-Net not having a optimal training configuration or that it is not used as intended where the input size equals the output size. Also, the CNN could be worse due to differences in this implementation of the CNN and the original implementation from Kleesiek et al. (2016).

5.3 Testing U-Net and the CNN on different configurations

Testing the U-Net and the CNN on different configurations showed that choosing the right configuration for the right task can speed up the training. The first run with a batch size of 4 for the CNN showed that the configuration using the Titan X was the fastest followed closely by the configuration using the GTX 1080. Using 1 or 2 Tesla P100 GPUs was the slowest, though using 2 Tesla GPUs was slightly faster compared to using 1. When the batch size was increased to 16 the Titan X configuration was still the fastest, but this time the GTX 1080 was the slowest configuration. The difference between using 1 or 2 Tesla P100 also became more prominent when using a bigger batch size with the configuration using 2 Tesla P100 being the fastest.

The reason for why the Titan X performed better than the Tesla P100 is most likely because of the higher clock speed of the GPU. It was faster compared to the GTX 1080 because of its much higher amount of CUDA cores. When the batch size was increased to 16 the amount of CUDA cores became more important because many more computations had to be done in parallel. Therefore, the GTX 1080 became the slowest configuration and the gap between using 1 and 2 Tesla P100 GPUs became much larger.

Training the U-Net with a batch size of 4 the configuration using the GTX 1080 was the slowest. The Titan X configuration was still the fastest, but the training time for the configurations using the Tesla P100's was closer compared to when the CNN was used. When the batch size of the U-Net was increased to 8 the configuration using 2 Tesla P100 was the fastest as figure 4.27 and table 4.21 shows. The GTX 1080 GPU could not handle a batch size of 8 for the U-Net

configuration as it ran out of memory when training. The P100 configuration and the Titan X configuration were able to train the U-Net because the GPUs have more memory available.

The reason for why the GTX 1080 was the slowest training the U-Net with batch size 4, but the second fastest when training the CNN with the same batch size, is because the U-Net has many more parameters. More parameters indicate more parallel computations and the clock speed of the GPU becomes less important compared to the amount of CUDA cores. Using 2 Tesla P100 was the fastest when training U-Net with batch size 8 because more parallel computations are done. This is the reason for why the configuration using 2 Tesla P100 was the fastest.

One thing to note is that the Nvidia P100 GPU is designed to be used with heavy load over time. If the GPU is used under heavy load for a long time the consumer options, such as the GTX 1080 and the Titan X, might break much earlier compared to high end GPUs like the Tesla P100.

This experiment demonstrates that choosing the right hardware configuration when training can speed up the process. The GPU is the most important since much of the processing is done on it. It is also the deciding factor if the deep learning architecture can be trained on the configuration. Since the architecture is loaded onto GPU and therefore the amount of memory a GPU has decides if the architecture can be trained on the configuration at all. The GPU memory also decides how big the batch size can be when training a network. Furthermore, the number of parameters in an architecture decides if the number of cores or the clock speed is the most important. This answers **RQ 4**.

5.4 Experiment on liver segmentation

With a dice score of 0.81 DeepMedic seems to be the only architecture capable of doing liver segmentation. The CNN got a dice score of 0.61 while the U-Net only predicted non-liver voxels for the whole image. The reason for the architecture having a high sensitivity and specificity is that only approximately 3% of the voxels in the CT image are liver compared to the brain MRI scans where the brain accounts for approximately 25% of the voxels. The very low amount of liver voxels present compared to non-liver voxels could also have caused this implementation of U-Net to perform badly when doing predictions. During training the U-Net tried to minimize the loss by predicting only non-liver voxels. A solution to this could be to sample input patches during training so that the majority of the patches are centered around the liver instead of the randomized approach that this implementation uses.

Figure 4.28 shows that one of the reasons for the low dice scores for the DeepMedic is false positive predictions in the lower part of the image. The predictions around the liver seems to be somewhat accurate with a few false positives and a few false negatives. As for the CNN it also had some false positives in the lower part of the image. In addition, it had many false negatives around the liver, further contributing to a lower dice score.

This experiment shows, at least for the CNN and DeepMedic, that the architectures have no problems on other medical image modalities. The bad scores in this experiment are most likely a result of liver segmentation being a harder task compared to skull stripping.

If some post-processing was done on the DeepMedic prediction to cut away some false positives the DeepMedic architecture could maybe get a fairly high dice score. This should not be overly difficult either, for example one solution could be to post-process the prediction so that only the largest connection of predicted positive voxels are kept. Since much of the image is essentially useless for the network when doing liver prediction, another solution would be to cut out parts of the image so that only the part where the liver is located is kept. This could also maybe help the CNN and the U-Net.

Finally, it should be added that the models were trained and tested on the same data for this experiment. The scores are therefore not really significant since the models could simply have overtrained on the data. However, this experiment also shows that liver segmentation is possible, at least with DeepMedic. Other 3D CNNs used for liver segmentation, like Lu et al. (2017), used post-processing to get better results. Figure 4.28, which shows DeepMedic's best prediction on the LiTS data, indicates that simple post-processing like cutting away irregularly placed falsely predicted liver voxels might make the final segmentation better. If this is done the scores for DeepMedic might become much better. **RQ 5** asked if the deep learning architectures could also be used to do liver segmentation. The answer to this is that it seems possible to do this with the CNN and DeepMedic, but not this implementation of U-Net.

5.5 Reflection

When implementing the U-Net in Keras a lot of mistakes were made at the start. Deep learning can be very hard to debug since the program will show no errors, but it can still produce unpredictable results because of mistakes in the code. Also, deep learning usually needs very good hardware to run. This was the case for all the architectures used in this thesis causing it to be impossible to do debugging on the laptop. For example during the implementation of U-Net one error in one line used for building the model lead to weeks spent on trying to figure out if the U-Net could do skull stripping at all. U-Net would train and predict just fine, but the predictions were useless. At one point another implementation of U-Net found on GitHub was tested to see if U-Net could do skull stripping. That implementation could do skull stripping which lead to debugging of the code in this implementation and the error was eventually found. The lesson in this is that it is important to search the code for errors and not only spend time on trying new things. Less time could have been wasted if time was first spent on looking at the code instead of trying other things.

At first it was assumed that a larger patch size for the U-Net would produce the best results. Every experiment was first done using patch size 64^3 for the U-Net which was the largest patch

size that could be used, any larger and the training time would be too long. A smaller patch size was tried with the U-Net and it lead to better results. Consequently, all of the experiments had to be repeated for the U-Net. Lesson learned is that it is very important to set up experiments were different configurations of the architecture are tried out.

In this master thesis the CNN and the U-Net was implemented in the same solution. This was not necessarily a bad, but it would have been cleaner to start from scratch with the implementation of U-Net. Using the skills learned when reimplementing the CNN from Kleesiek et al. (2016) in Keras the code for U-Net could have been better and perhaps less time would have been used debugging it at the start.

Using NTNU's EPIC was very easy after getting used to it. EPIC enables the user to train many models at the same time, which is perfect if the user wants to explore new deep learning methods or test modifications on existing implementations. If learning how to use EPIC happened sooner many more experiments could have been done and perhaps the CNN and U-Net could produce better results.

Another point is that no regularization methods were explored for the U-Net, or for that matter the CNN. It is trivial to add regularization in Keras, but it did not seem necessary since this was not used in the CNN from Kleesiek et al. (2016). The data augmentation used in Kleesiek et al. (2016) should have a mitigating effect on overfitting, but other regularization methods should perhaps have been tried out as well. If this was done a larger patch size for the U-Net could perhaps have been used since the problem with the larger patch sizes was overfitting.

Chapter 6

Conclusion and future work

6.1 Conclusion

In this thesis various experiments with different deep learning architectures have been completed. The CNN from Kleesiek et al. (2016), U-Net from Çiçek et al. (2016) and DeepMedic from Kamnitsas et al. (2016) was used to train on data from OASIS, LBPA40 and data from the St. Olavs hospital. The first research question asked how important it was for different deep learning architectures to train on data from the same source before performing skull stripping. This was shown to be very important for all the architectures used in this thesis. No architecture could provide satisfactory results on any data set that they had not been trained on.

An experiment where the architectures were trained and tested on the same number of images from each data set was done to answer the second research question. This was shown to be detrimental when testing on the data set that was most variable and that originally had the most data. For the data set with the least amount of data a small improvement could be seen on two architectures: the CNN and DeepMedic.

The third research question asked if training on images with the same voxel size is important when performing skull stripping. This was shown to be important for the St. Olavs data set since it was the most irregular data set. The CNN and the U-Net had the most problems with not resampled data, while DeepMedic had less problems. For the LBPA40 and the OASIS data set this was shown to not be as an important factor because they were not as irregular.

Furthermore, this thesis has explored different deep learning solutions that the St. Olavs hospital can use for skull stripping. DeepMedic was found to be the best solution that was tested.

Research question four asked how different hardware configurations impact the training time of deep learning methods. The experiments done showed that choosing the right configuration can speed up the training time. If an architecture has many parameter and uses a large batch size the amount of CUDA cores becomes more important when training, but if the architecture has less

parameters and uses a smaller batch size the clock speed of the GPU becomes more important.

Lastly the architectures were tested on liver segmentation data. DeepMedic showed promising results, but it produced many spurious false positives. The CNN also had spurious false positive and it also had many false negatives in its prediction. This implementation of U-Net was shown to not be able to do liver segmentation.

6.2 Future work

CRF can be used to make better predictions. Originally, DeepMedic (Kamnitsas et al., 2017) used CFR to do post-processing on its tumor predictions. Using CRF could lead to better brain mask predictions.

The training scheme for this implementation of U-Net was the same as the one used for the CNN. The training scheme should be reimplemented so that it is tailored to the U-Net. Also, a larger patch size for the U-Net could be used if more regularization techniques are added to the implementation. This should not be difficult to investigate.

Some results could be improved if some post-processing steps were added. For example to fill in holes in U-Net's predictions and cutting away falsely predicted brain voxels in some predictions.

The capsule network by Sabour et al. (2017) should perhaps be looked more into. As of now, there has not been published much work on this architecture since it is so new, but LaLonde and Bagci (2018) showed that the architecture can successfully be used for medical imaging segmentation. The paper did not extend the capsule network to handle 3D data, but this could be explored further.

Bibliography

Akkus, Z., Galimzianova, A., Hoogi, A., Rubin, D. L., Erickson, B. J., 2017. Deep Learning for Brain MRI Segmentation: State of the Art and Future Directions. Journal of Digital Imaging 30 (4), 449–459.

Ben-Cohen, A., Diamant, I., Klang, E., Amitai, M., Greenspan, H., 2016. Fully Convolutional Network for Liver Segmentation and Lesions Detection. In: Carneiro, G., Mateus, D., Peter, L., Bradley, A., Tavares, J. M. R. S., Belagiannis, V., Papa, J. P., Nascimento, J. C., Loog, M., Lu, Z., Cardoso, J. S., Cornebise, J. (Eds.), Deep Learning and Data Labeling for Medical Applications. Springer International Publishing, Cham, pp. 77–85.

Buckner, R. L., Head, D., Parker, J., Fotenos, A. F., Marcus, D., Morris, J. C., Snyder, A. Z., 2004. A unified approach for morphometric and functional data analysis in young, old, and demented adults using automated atlas-based head size normalization: Reliability and validation against manual measurement of total intracranial volume. NeuroImage 23 (2), 724–738.

Casamitjana, A., Puch, S., Aduriz, A., Vilaplana, V., 2016. 3D Convolutional Neural Networks for Brain Tumor Segmentation: A Comparison of Multi-resolution Architectures. In: Crimi, A., Menze, B., Maier, O., Reyes, M., Winzeck, S., Handels, H. (Eds.), Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries. Springer International Publishing, Cham, pp. 150–161.

Çiçek, Ö., Abdulkadir, A., Lienkamp, S. S., Brox, T., Ronneberger, O., 2016. 3D U-net: Learning dense volumetric segmentation from sparse annotation. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 9901 LNCS, 424–432.

COX, R. W., 1996. AFNI : Software for Analysis and Visualization of Functional Magnetic Resonance Neuroimages. COMPUTERS AND BIOMEDICAL RESEARCH 173 (29), 162–173.

de Boer, R., Vrooman, H. A., Ikram, M. A., Vernooij, M. W., Breteler, M. M. B., van der Lugt, A., Niessen, W. J., 2010. Accuracy and reproducibility study of automatic MRI brain tissue

segmentation methods. NeuroImage 51 (3), 1047–1056.
URL http://dx.doi.org/10.1016/j.neuroimage.2010.03.012

Dumoulin, V., Visin, F., 2016. A guide to convolution arithmetic for deep learning, 1–28.
URL http://arxiv.org/abs/1603.07285

Eide, Ø. A., 2017. Skull stripping using deep neural networks. Ph.D. thesis, NTNU.
URL https://github.com/oysteiae/FordypningsProsjekt

Eskildsen, S. F., Coupé, P., Fonov, V., Manjón, J. V., Leung, K. K., Guizard, N., Wassef, S. N., Østergaard, L. R., Collins, D. L., 2012. BEaST: Brain extraction based on nonlocal segmentation technique. NeuroImage 59 (3), 2362–2373.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press.

He, K., Zhang, X., Ren, S., Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. Proceedings of the IEEE International Conference on Computer Vision 2015 Inter, 1026–1034.

Iglesias, J. E., Liu, C. Y., Thompson, P. M., Tu, Z., 2011. Robust brain extraction across datasets and comparison with publicly available methods. IEEE Transactions on Medical Imaging 30 (9), 1617–1634.

Ioffe, S., Szegedy, C., 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. CoRR abs/1502.0.
URL http://arxiv.org/abs/1502.03167

Kamnitsas, K., Ferrante, E., Parisot, S., Ledig, C., Nori, A. V., Criminisi, A., Rueckert, D., Glocker, B., 2016. DeepMedic for Brain Tumor Segmentation, 138–149.
URL http://link.springer.com/10.1007/978-3-319-55524-9{_}14

Kamnitsas, K., Ledig, C., Newcombe, V. F., Simpson, J. P., Kane, A. D., Menon, D. K., Rueckert, D., Glocker, B., 2017. Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation. Medical Image Analysis 36, 61–78.
URL http://dx.doi.org/10.1016/j.media.2016.10.004

Ker, J., Wang, L., Rao, J., Lim, T., 2018. Deep Learning Applications in Medical Image Analysis. IEEE Access, 1–1.
URL http://ieeexplore.ieee.org/document/8241753/

Kim, H., Nam, H., Jung, W., Lee, J., apr 2017. Performance analysis of CNN frameworks for GPUs. In: 2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). pp. 55–64.

Kleesiek, J., Urban, G., Hubert, A., Schwarz, D., Maier-Hein, K., Bendszus, M., Biller, A., 2016. Deep MRI brain extraction: A 3D convolutional neural network for skull stripping.

NeuroImage 129, 460–469.
URL bi

Klein, A., Ghosh, S. S., Avants, B., Yeo, B. T., Fischl, B., Ardekani, B., Gee, J. C., Mann, J. J., Parsey, R. V., 2010. Evaluation of volume-based and surface-based brain image registration methods. NeuroImage 51 (1), 214–220.
URL http://dx.doi.org/10.1016/j.neuroimage.2010.01.091

LaLonde, R., Bagci, U., 2018. Capsules for Object Segmentation (Midl), 1–9.
URL http://arxiv.org/abs/1804.04241

Ling, H., Kevin Zhou, S., Zheng, Y., Georgescu, B., Suehling, M., Comaniciu, D., 2008. Hierarchical, learning-based automatic liver segmentation. 26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR.

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., van der Laak, J. A. W. M., van Ginneken, B., Sánchez, C. I., 2017. A Survey on Deep Learning in Medical Image Analysis. CoRR (1995).
URL http://arxiv.org/abs/1702.05747{%}0Ahttp://dx.doi.org/10.1016/j.media.2017.07.005

Lu, F., Wu, F., Hu, P., Peng, Z., Kong, D., feb 2017. Automatic 3D liver location and segmentation via convolutional neural network and graph cut. International Journal of Computer Assisted Radiology and Surgery 12 (2), 171–182.
URL https://doi.org/10.1007/s11548-016-1467-3

MacDonald, D., Kabani, N., Avis, D., Evans, A. C., 2000. Automated 3-D extraction of inner and outer surfaces of cerebral cortex from MRI. NeuroImage 12 (3), 340–356.

Marcus, D. S., Wang, T. H., Parker, J., Csernansky, J. G., Morris, J. C., Buckner, R. L., 2007. Open Access Series of Imaging Studies (OASIS): Cross-sectional MRI data in young, middle aged, nondemented, and demented older adults. Journal of Cognitive Neuroscience 19 (9), 1498–1507.

Masci, J., Giusti, A., Ciresan, D. C., Fricout, G., Schmidhuber, J., 2013. A Fast Learning Algorithm for Image Segmentation with Max-Pooling Convolutional Networks. CoRR abs/1302.1.
URL http://arxiv.org/abs/1302.1690

Milletari, F., Navab, N., Ahmadi, S. A., 2016. V-Net: Fully convolutional neural networks for volumetric medical image segmentation. Proceedings - 2016 4th International Conference on 3D Vision, 3DV 2016, 565–571.

Nie, D., Wang, L., Gao, Y., Sken, D., 2016. Fully convolutional networks for multi-modality isointense infant brain image segmentation. Proceedings - International Symposium on Biomedical Imaging 2016-June, 1342–1345.

Nielsen, M., 2015. Neural Networks and Deep Learning. Determination press.
URL http://neuralnetworksanddeeplearning.com/index.html

Perona, P., Malik, J., 1990. Scale-space and edge detection using anisotropic diffusion. IEEE Transactions on Pattern Analysis and Machine Intelligence 12 (7), 629–639.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=56205

Preston, D. C., 2006. Magnetic Resonance Imaging (MRI) of the Brain and Spine: Basics.
URL http://casemed.case.edu/clerkships/neurology/WebNeurorad/MRIBasics.htm

Rohlfing, T., feb 2012. Image Similarity and Tissue Overlaps as Surrogates for Image Registration Accuracy: Widely Used but Unreliable. IEEE Transactions on Medical Imaging 31 (2), 153–163.

Ronneberger, O., Fischer, P., Brox, T., 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. CoRR abs/1505.0.
URL http://arxiv.org/abs/1505.04597

Sabour, S., Frosst, N., Hinton, G. E., 2017. Dynamic Routing Between Capsules (Nips).
URL http://arxiv.org/abs/1710.09829

Sebastian, R., 2016. Monte carlo EM for missing covariates in parametric regression models. CoRR.
URL http://arxiv.org/abs/1609.04747

Ségonne, F., Dale, A. M., Busa, E., Glessner, M., Salat, D., Hahn, H. K., Fischl, B., 2004. A hybrid approach to the skull stripping problem in MRI. NeuroImage 22 (3), 1060–1075.

Shattuck, D. W., Mirza, M., Adisetiyo, V., Hojatkashani, C., Salamon, G., Narr, K. L., Poldrack, R. A., Bilder, R. M., Toga, A. W., 2008. Construction of a 3D probabilistic atlas of human cortical structures. NeuroImage 39 (3), 1064–1080.

Simonyan, K., Zisserman, A., 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition, 1–14.
URL http://arxiv.org/abs/1409.1556

Smith, S. M., 2002. Fast robust automated brain extraction. Human Brain Mapping 17 (3), 143–155.

Speier, W., Iglesias, J. E., El-Kara, L., Tu, Z., Arnold, C., 2011. Robust skull stripping of clinical glioblastoma multiforme data. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 6893 LNCS (PART 3), 659–666.

Thompson, P. M., Mega, M. S., Woods, R. P., Zoumalan, C. I., Lindshield, C. J., Blanton, R. E., Moussai, J., Holmes, C. J., Cummings, J. L., Toga, A. W., 2001. Cortical Change in

Alzheimer's Disease Detected with a Disease-specific Population-based Brain Atlas. Cerebral Cortex 11 (1), 1–16.
URL http://dx.doi.org/10.1093/cercor/11.1.1

Tosun, D., Rettmann, M. E., Naiman, D. Q., Resnick, S. M., Kraut, M. A., Prince, J. L., 2006. Cortical reconstruction using implicit surface evolution: Accuracy and precision analysis. NeuroImage 29 (3), 838–852.

Yang, D., Xu, D., Zhou, S. K., Georgescu, B., Chen, M., Grbic, S., Metaxas, D., Comaniciu, D., 2017. Automatic liver segmentation using an adversarial image-to-image network. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 10435 LNCS, 507–515.

Zeiler, M. D., Krishnan, D., Taylor, G. W., Fergus, R., 2010. Deconvolutional networks. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2528–2535.

Zhang, Y., Brady, M., Smith, S., jan 2001. Segmentation of brain MR images through a hidden Markov random field model and the expectation-maximization algorithm. IEEE Transactions on Medical Imaging 20 (1), 45–57.

Zhao, L., Jia, K., 2016. Deep Feature Learning with Discrimination Mechanism for Brain Tumor Segmentation and Diagnosis. Proceedings - 2015 International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIH-MSP 2015, 306–309.

Zhao, X., Wu, Y., Song, G., Li, Z., Fan, Y., Zhang, Y., 2016. Brain tumor segmentation using a fully convolutional neural network with conditional random fields. Vol. 10154 LNCS.

Appendix

Appendix A

Experiments with data sets and models

The accuracy graph presented for the U-Net and the CNN show averaged accuracies after 100 update steps. For DeepMedic the accuracy is plotted using a function that is included in the DeepMedic implementation.

A.1 Networks trained and tested on one data set

A.1.1 CNN

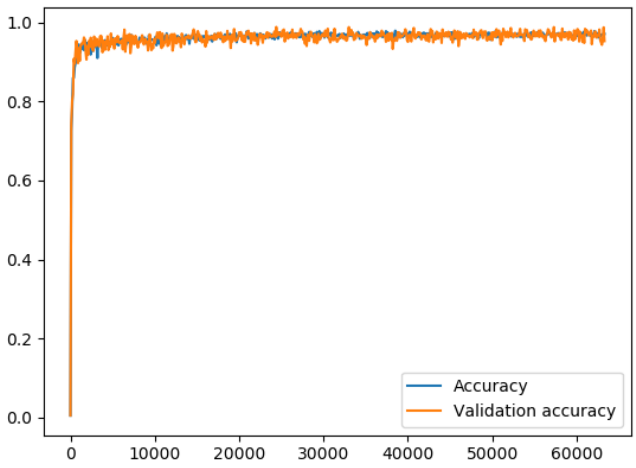


Figure A.1: *Accuracy graph CNN only OASIS.*

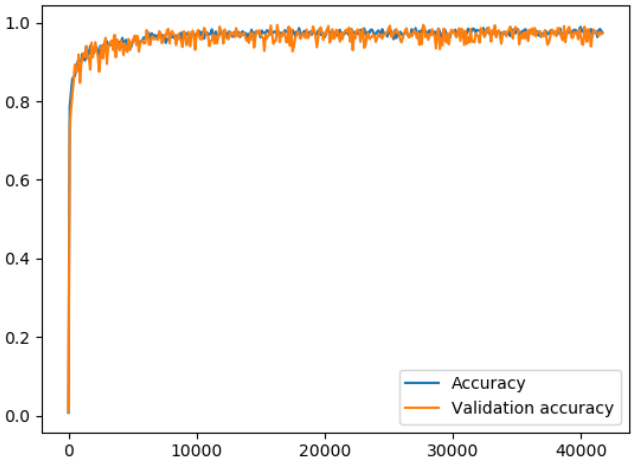


Figure A.2: *Accuracy graph CNN only St. Olavs.*

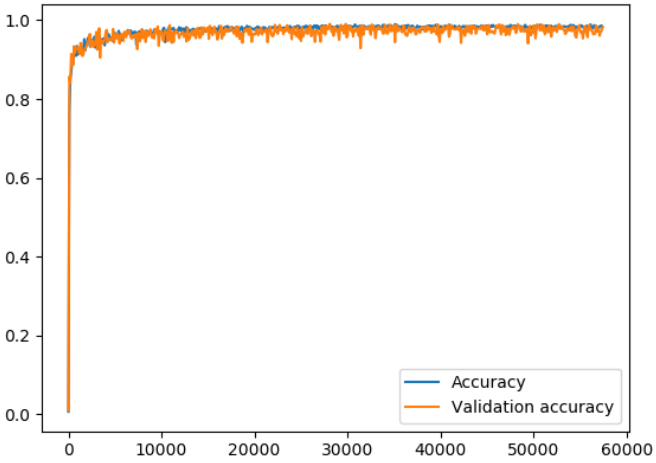


Figure A.3: *Accuracy graph CNN only LBPA40.*

A.1.2 U-Net

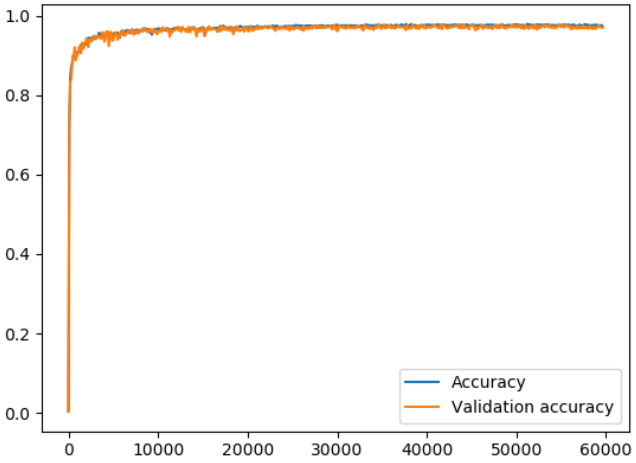


Figure A.4: *Accuracy graph U-Net only OASIS.*

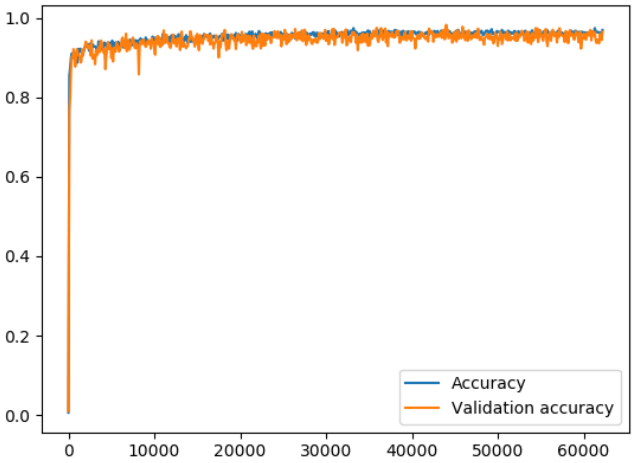


Figure A.5: *Accuracy graph U-Net only St. Olavs.*

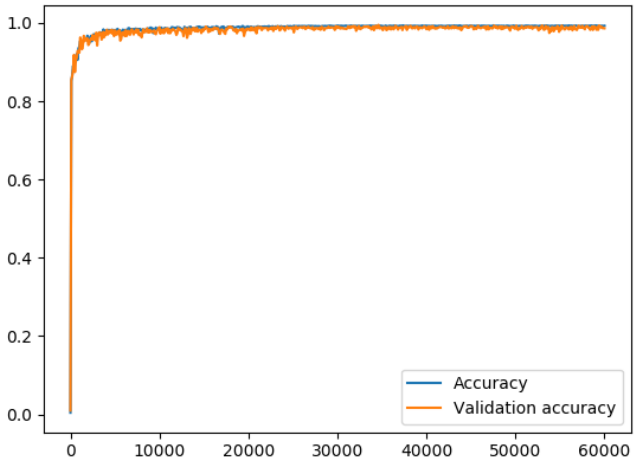


Figure A.6: *Accuracy graph U-Net only LBPA40.*

A.1.3 DeepMedic

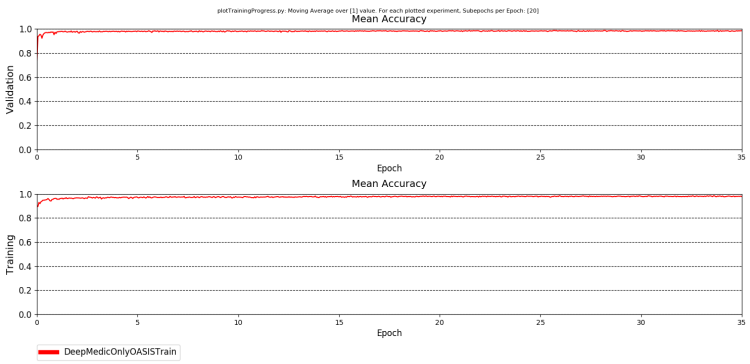


Figure A.7: *Accuracy graph DeepMedic only OASIS.*

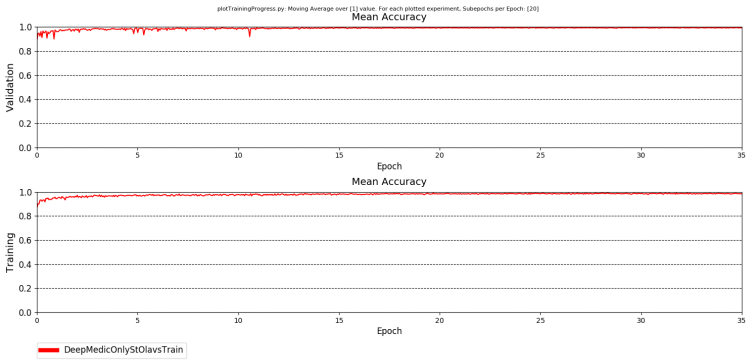


Figure A.8: *Accuracy graph DeepMedic only St. Olavs.*

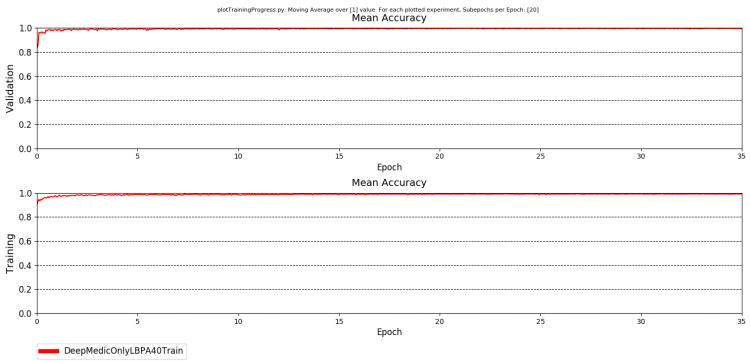


Figure A.9: *Accuracy graph DeepMedic only LBPA40.*

A.2 Networks trained on two data sets tested on remaining

A.2.1 CNN

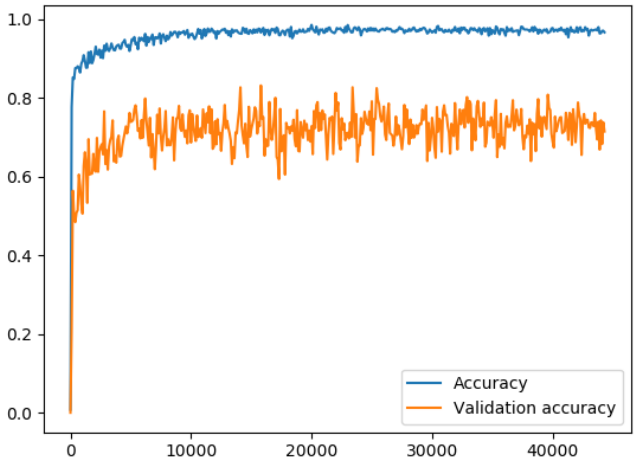


Figure A.10: *Accuracy graph CNN trained on LBPA40 and St. Olavs validated on OASIS.*

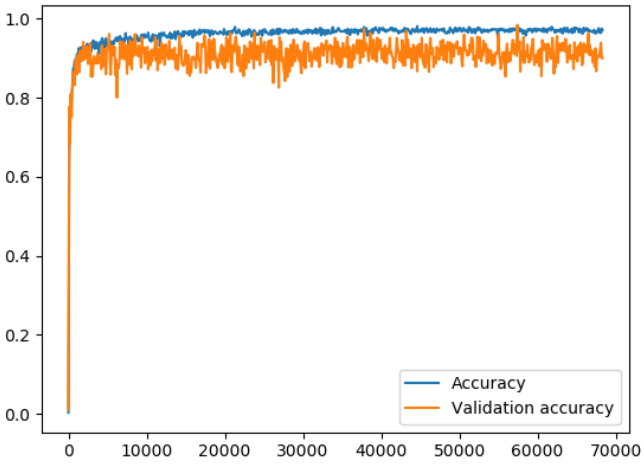


Figure A.11: *Accuracy graph CNN trained on LBPA40 and OASIS validated on St. Olavs.*

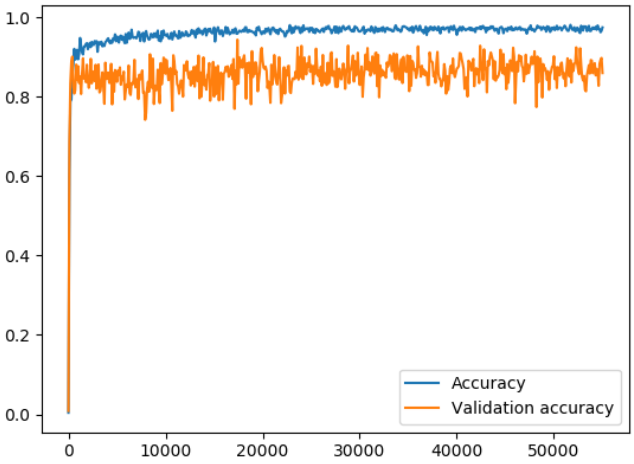


Figure A.12: *Accuracy graph CNN trained on OASIS and St. Olavs validated on LBPA40.*

A.2.2 U-Net

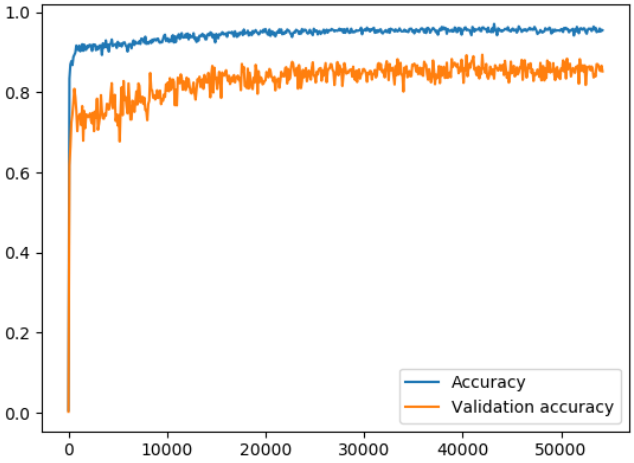


Figure A.13: *Accuracy graph U-Net trained on LBPA40 and St. Olavs validated on OASIS.*

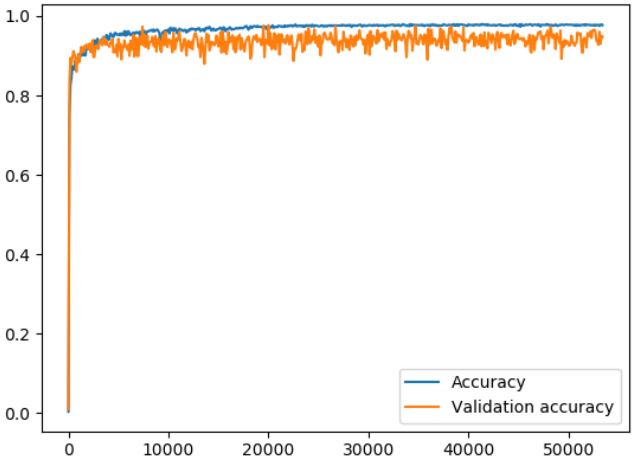


Figure A.14: *Accuracy graph U-Net trained on LBPA40 and OASIS validated on St. Olavs.*

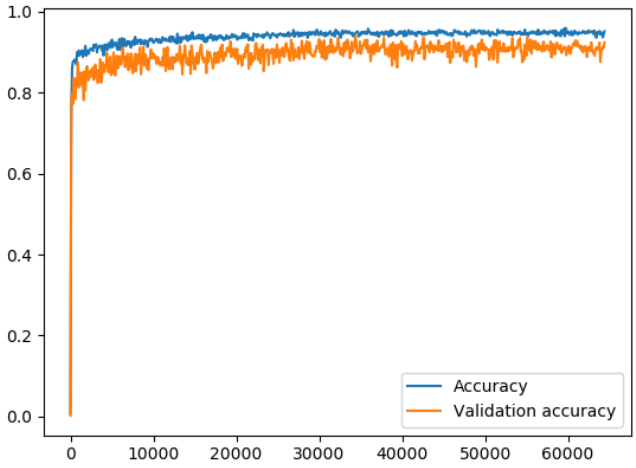


Figure A.15: *Accuracy graph U-Net trained on OASIS and St. Olavs validated on LBPA40.*

A.2.3 DeepMedic

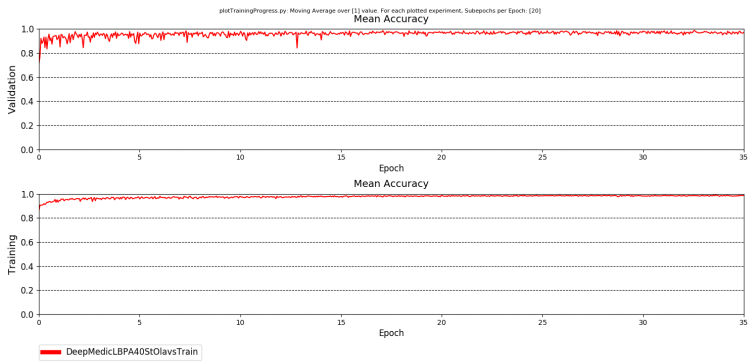


Figure A.16: *Accuracy graph DeepMedic trained on LBPA40 and St. Olavs validated on OASIS.*

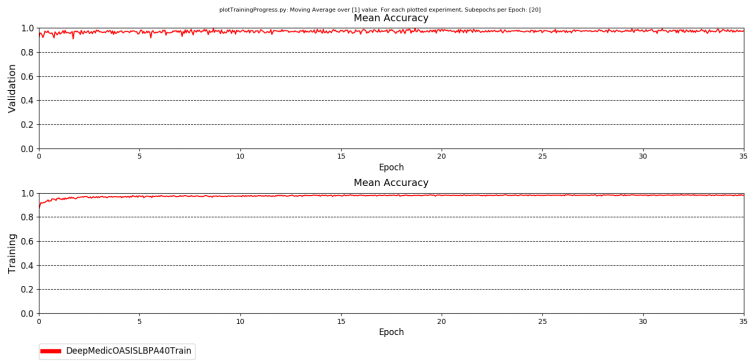


Figure A.17: *Accuracy graph DeepMedic trained on LBPA40 and OASIS validated on St. Olavs.*

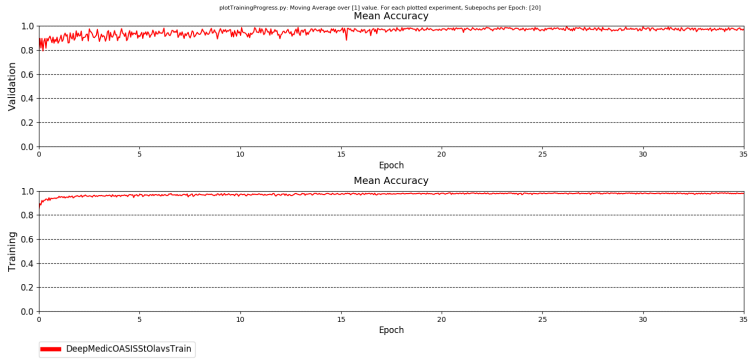


Figure A.18: *Accuracy graph DeepMedic trained on OASIS and St. Olavs validated on LBPA40.*

A.3 Networks trained on all the data

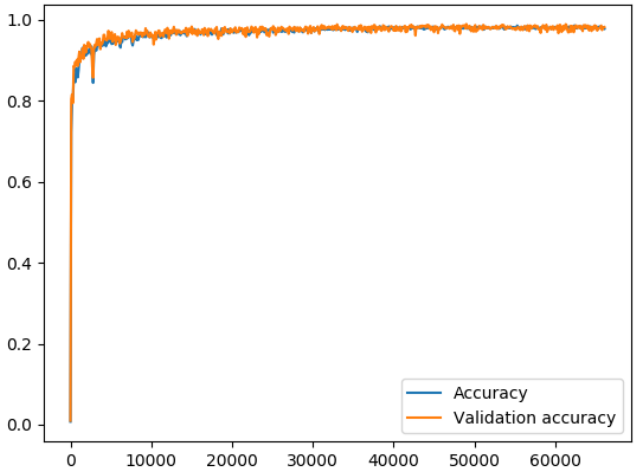


Figure A.19: *Accuracy and validation accuracy for U-Net trained on all data.*

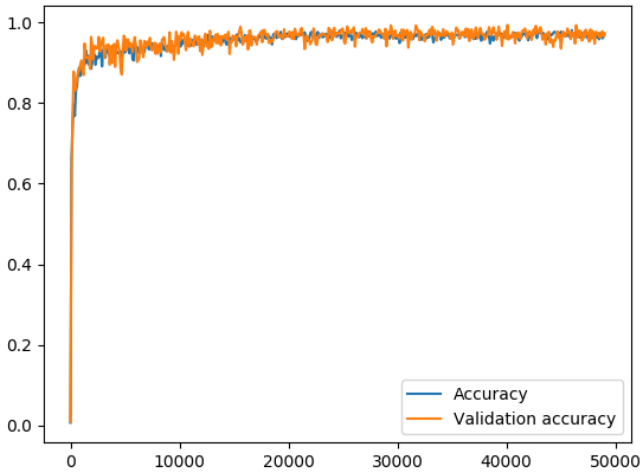


Figure A.20: *Accuracy and validation accuracy for the CNN trained on all data.*

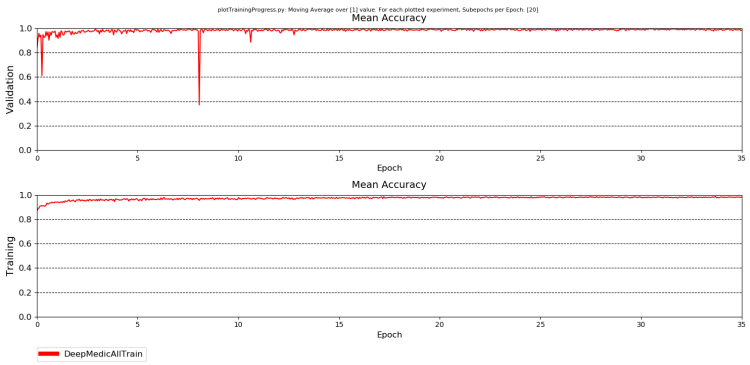


Figure A.21: *Accuracy and validation accuracy for the CNN trained on all data.*

A.4 Networks trained on equal amounts of data from each data set

A.4.1 CNN

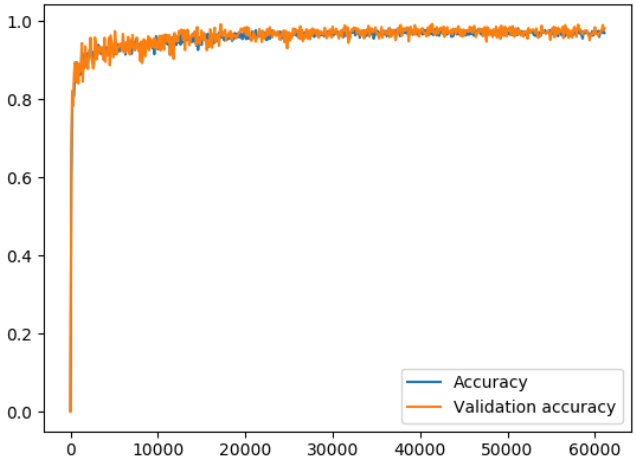


Figure A.22: *Accuracy graph CNN trained on equal amounts of data.*

A.4.2 U-Net

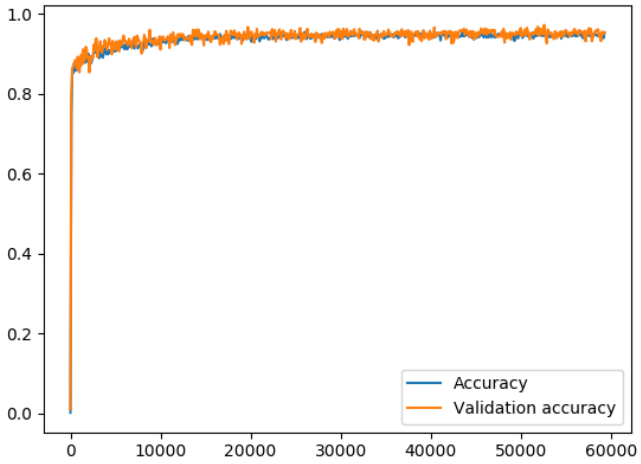


Figure A.23: *Accuracy graph U-Net trained on equal amounts of data.*

A.4.3 DeepMedic

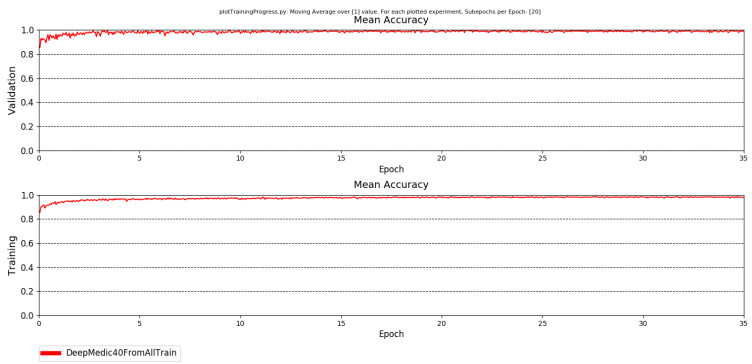


Figure A.24: *Accuracy graph DeepMedic trained on equal amounts of data.*

A.5 Network trained on not resampled data

A.5.1 CNN

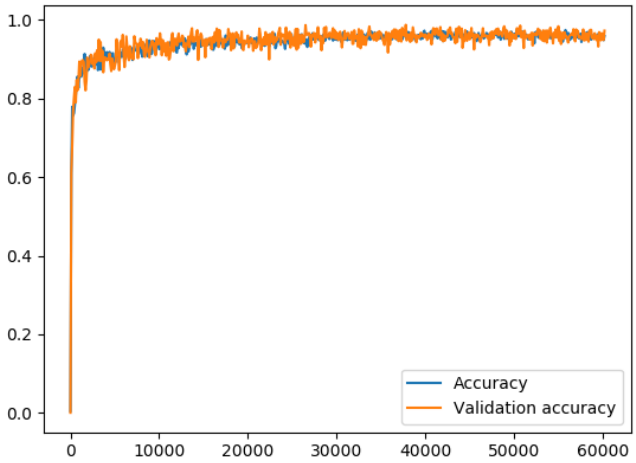


Figure A.25: *Accuracy graph CNN trained on not resampled data.*

A.5.2 U-Net

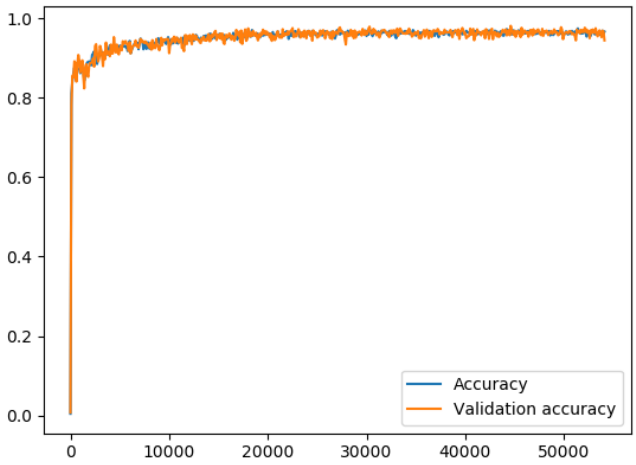


Figure A.26: *Accuracy graph U-Net trained on not resampled data.*

A.5.3 DeepMedic

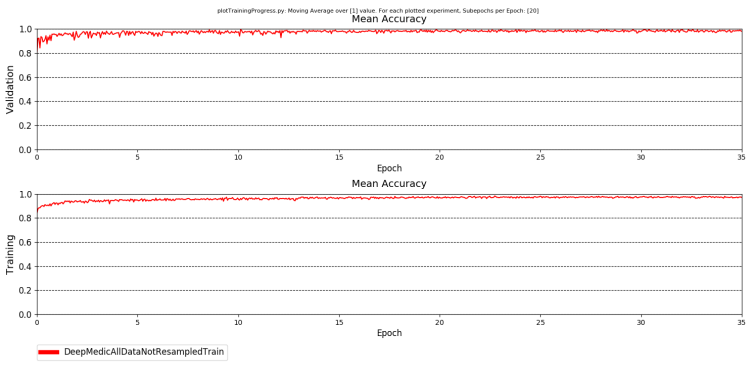


Figure A.27: *Accuracy graph DeepMedic trained on not resampled data.*

Appendix B

Feature maps

This appendix chapter shows feature maps for the first 4 layers of the different architectures. The feature maps where produced from models trained and tested on data from OASIS and LBPA40 as well as data from the St. Olavs hospital.

B.1 U-Net

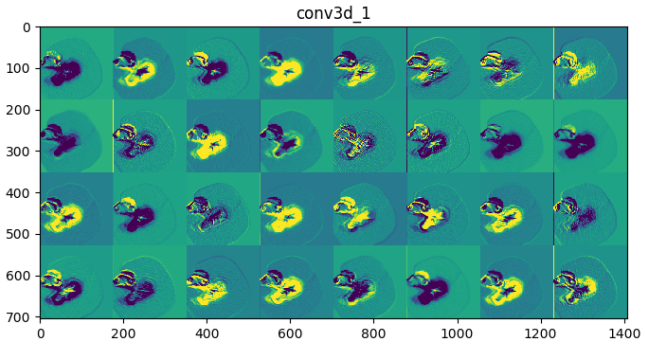


Figure B.1: *First convolutional layer of U-Net.*

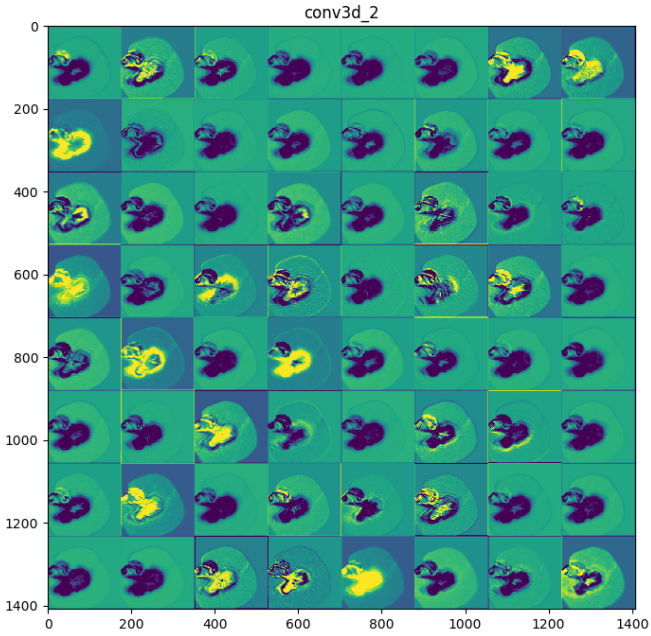


Figure B.2: *Second convolutional layer of U-Net.*

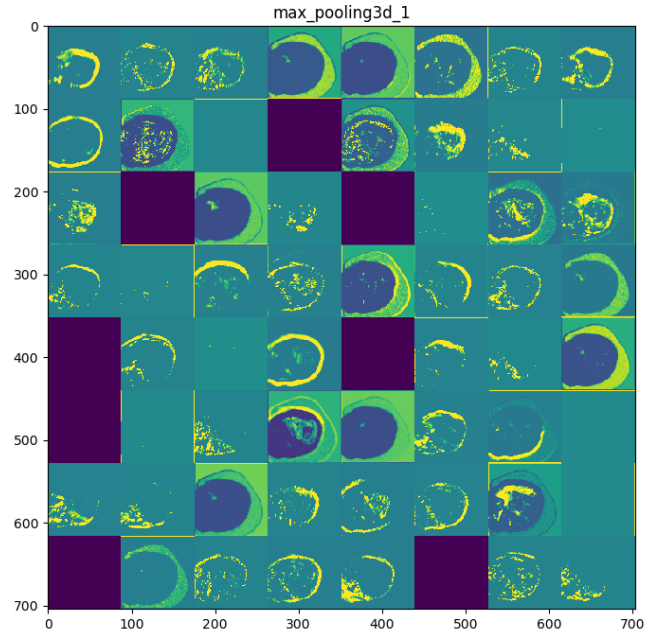


Figure B.3: *First max-pooling layer of U-Net.*

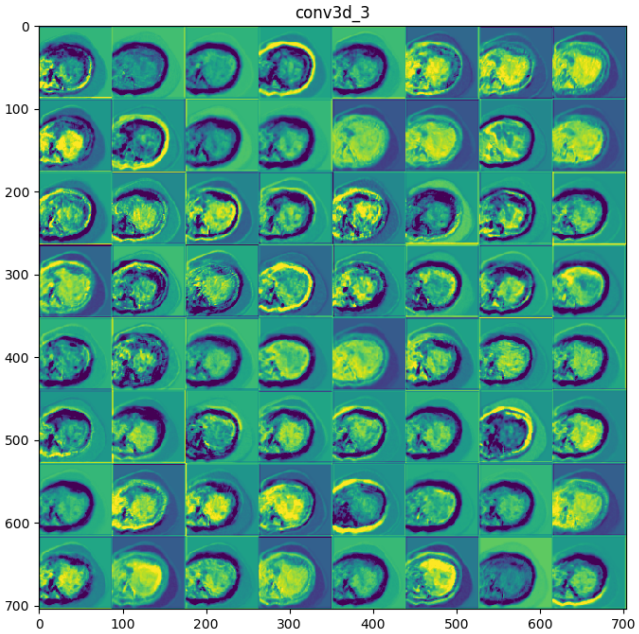


Figure B.4: *Third convolutional layer of U-Net.*

B.2 CNN

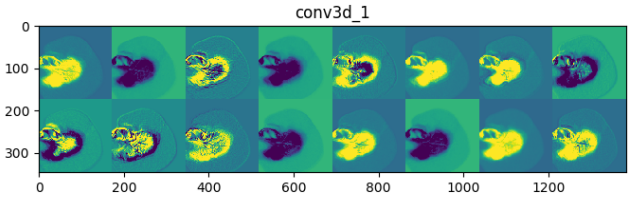


Figure B.5: *First convolutional layer of the CNN.*

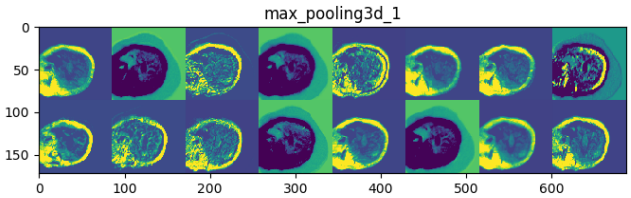


Figure B.6: *First max pooling layer of the CNN.*

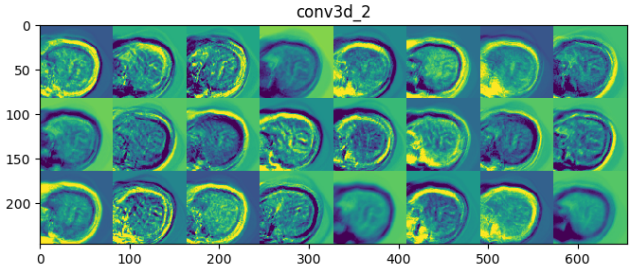


Figure B.7: *Second convolutional layer of the CNN.*

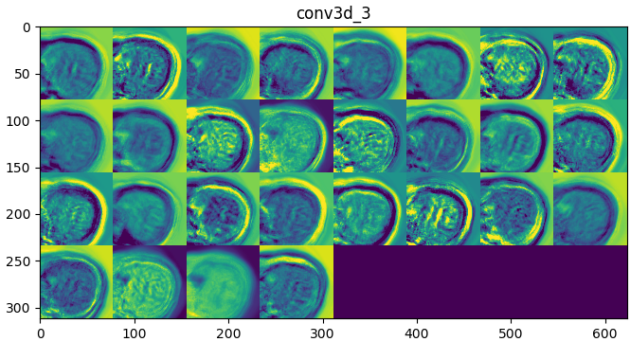


Figure B.8: *Third convolutional layer of the CNN.*

B.3 DeepMedic

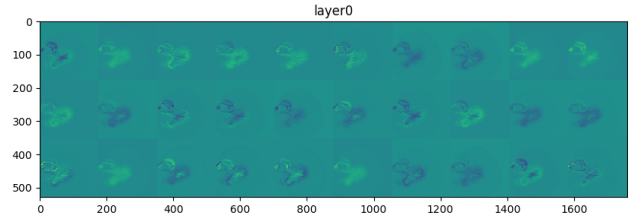


Figure B.9: *Pathway 0 DeepMedic convolutional layer 1.*

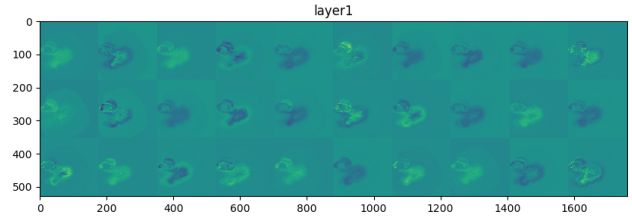


Figure B.10: *Pathway 0 DeepMedic convolutional layer 2.*

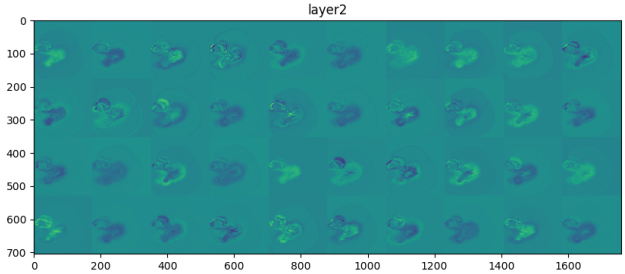


Figure B.11: *Pathway 0 DeepMedic convolutional layer 3.*

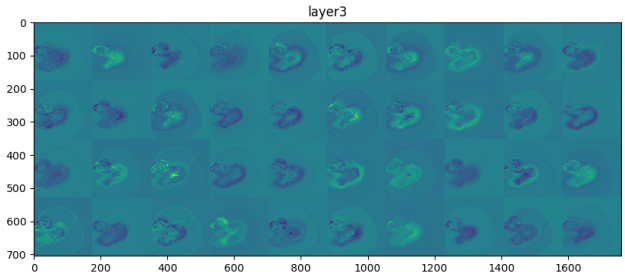


Figure B.12: *Pathway 0 DeepMedic convolutional layer 4.*

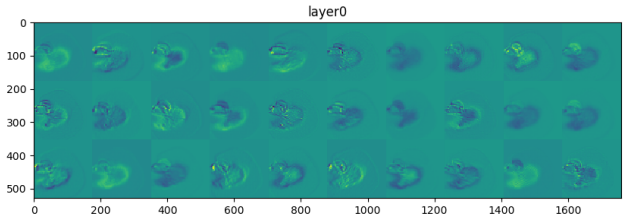


Figure B.13: *Pathway 1 DeepMedic convolutional layer 1.*

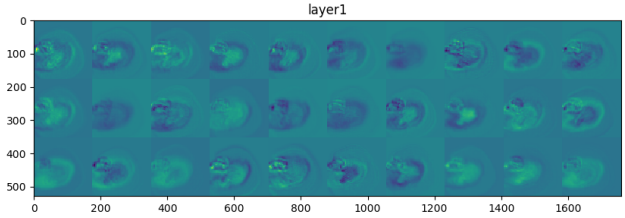


Figure B.14: *Pathway 1 DeepMedic convolutional layer 2.*

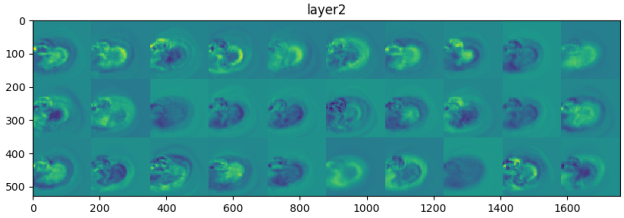


Figure B.15: *Pathway 1 DeepMedic convolutional layer 3.*

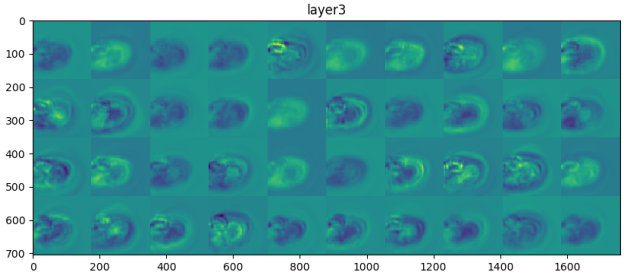


Figure B.16: *Pathway 1 DeepMedic convolutional layer 4.*