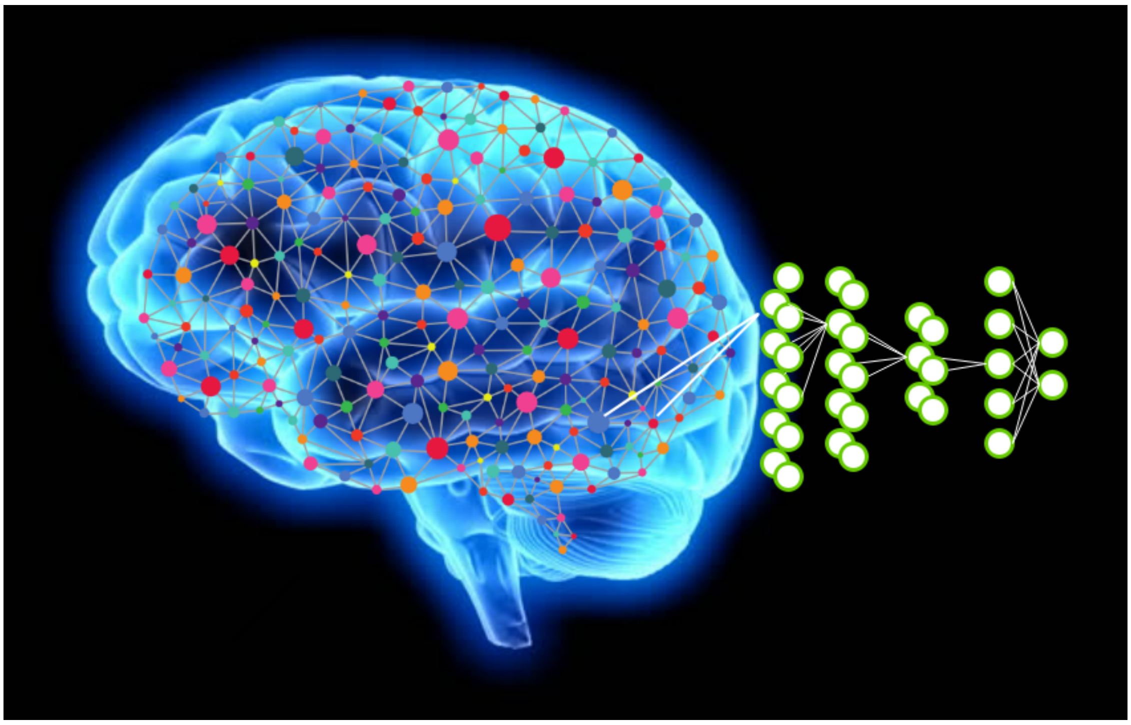
CHALMERS
UNIVERSITY OF TECHNOLOGY



Deep Learning Methods for MRI Brain Image Analysis:

3D Convolutional Neural Networks for Alzheimer's Disease detection and Brain Tumor classification

Master's thesis in Complex Adaptive Systems

Mahmood Nazari and Karl Bäckström

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2017, EX085/2017

Master's thesis 2017

Deep Learning Methods for MRI Brain Image Analysis:

3D Convolutional Neural Networks for Alzheimer's Disease detection and Brain Tumor classification

Mahmood Nazari and Karl Bäckström

Supervisor and Examiner: Prof. Irene Yu-Hua Gu,
Dept. of Electrical Engineering, Chalmers Univ. of Technology

Department of Electrical Engineering
Chalmers University of Technology
Göteborg, Sweden 2017, EX085/2017

Deep Learning Methods for MRI Brain Image Analysis
3D Convolutional Neural Networks for Alzheimer's Disease detection and Brain Tumor classification
Mahmood Nazari & Karl Bäckström

Master Thesis EX085/2017
Department of Electronic Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of a 3D brain image being analyzed by a neural network.

Typeset in LaTeX
Gothenburg, Sweden 2017

Deep Learning Methods for MRI Brain Image Analysis
3D Convolutional Neural Networks for Alzheimer's Disease detection and Brain Tumor classification
Mahmood Nazari & Karl Bäckström
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In this thesis, we investigate deep learning methods for medical image analysis, in particular brain disease detection and classification. We attempt to solve two problems, namely (1) Alzheimer's disease (AD) detection and (2) Brain tumor (BT) classification into high-grade and low-grade gliomas. Efficient and accurate diagnosis of AD is essential for initiating treatment at an early stage, increasing the chances of postponing the irreversible neurodegenerative effects of the disease. To this end, many studies have investigated image processing and machine learning methods for computer-aided-diagnosis (CAD) of AD. In the first part of our work, we continue along this line, and propose a simple 3D CNN architecture for feature extraction and classification of pre-processed T1-weighted MRI brain images for AD detection. The dataset used for training and evaluation consists of 340 subjects, and was obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI). The resulting test performance on unseen data is a classification accuracy of 98.74%, with a sensitivity (AD detection rate) of 100% which is comparable with the current state of the art. In this thesis, we thoroughly investigate the effect of some parameters involved in the CNN training process, such as dropout probability and learning rate. We also show the impact of data pre-processing on the end performance, by comparing several different pre-processing pipelines. For the second part of our work, namely brain tumor classification, we employ a similar method using CNNs, but using a different architecture. The dataset was obtained from the MICCAI BRaTS competition 2017 and consists of 285 subjects. Our preliminary study on brain tumor classification has shown promising results with a classification accuracy of 85.96%, but further work is required for a more in-depth study of the topic.

Keywords: Machine Learning, Deep Learning, Artificial Intelligence, Medical Imaging, Alzheimer's Disease, Convolutional Neural Network, Brain Tumor, High Grade Gliomas, Low Grade Gliomas, MRI

Acknowledgements

We would like to express our deepest gratidue to friends and family for their support and encouragement during our work on this thesis.

Also, we thank our supervisor and examiner Irene Yu-Hua Gu of the Electronic Engineering department at Chalmers University of Technology for all her help during this thesis, and for providing the necessary hardware without which this work would not have been possible.

Mahmood Nazari & Karl Bäckström, Göteborg, August 2017

Contents

1 Introduction

1.1 Alzheimer's Disease

Alzheimer's disease (AD) is a chronic neurodegenerative disease which is presently incurable, the cause of which is not yet understood. As of 2015, roughly 30 million people around the world suffer from AD [1]. Some of its symptoms consist of disorientation, language difficulties, memory loss and mood swings. The average life expectancy after onset is about 3 to 9 years and the disease progressively worsens the patient's quality of life. With the aging population, efforts have been made to find better treatments for this disease. However, attempts to develop more efficient drugs have seldom been successful [2]. For this reason, more focus has shifted towards finding more efficient methods for early diagnosis [3] of the disease, and postponing the irreversible damage caused by it.

Currently, the diagnosis of AD is done in a clinical setting by observing the progression of the symptoms of dementia. This implies that the patient must show signs of gradual mental deterioration, either by direct observation of the doctor or reported by family members or close friends. Apart from the subjective nature of clinical methods, theoretical approaches are also of importance. For example, the detection of specific biomarkers in the patient's cerebrospinal fluid (CSF) [3]. Retrieval of the CSF is done through an invasive method, which could be risky. Therefore it seems advantageous to look at non invasive methods such as imaging techniques.

Advanced medical imaging techniques such as Computed Tomography (CT), Magnetic Resonance Imaging (MRI), Single-Photon Emission Computed Tomography (SPECT), Position Emission Tomography (PET) are currently used to assist in the diagnostic process [4] by manual inspection of a neurologist. This can give an indication regarding the correct diagnose, as the difference between a scan of a healthy person and one diagnosed with AD is clear in many cases (see Figure 1.3). Still, this requires a medical expert to manually inspect the image, which is time consuming, and might give varying outcomes depending on the experience of the examiner. In order to quickly and accurately determine the diagnosis, it is desirable to have system for Computer-aided diagnosis (CAD). For example, a system for automatic analysis of medical images could provide a medical doctor with markers giving indications of the correct diagnosis. This could potentially reduce the time and cost for the diagnostic process, as well as patient discomfort. To this end, we will in this project investigate the possibility of applying methods of artificial intelligence and machine learning to develop a system that can automatically, without the requirement for manual inspection, detect AD using one MRI scan of a patient.

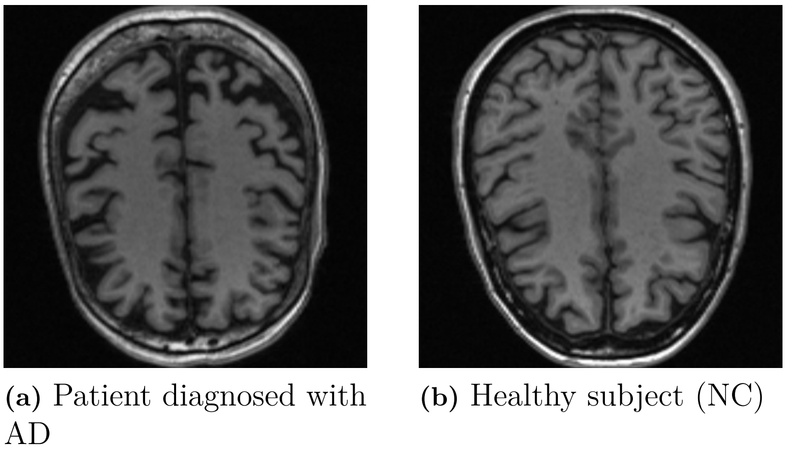

(a) Patient diagnosed with AD
(b) Healthy subject (NC)

Figure 1.1: Visualization of the difference between the MRI scan of a patient diagnosed with Alzheimer's disease, compared to a healthy, elderly person in original MRI images.

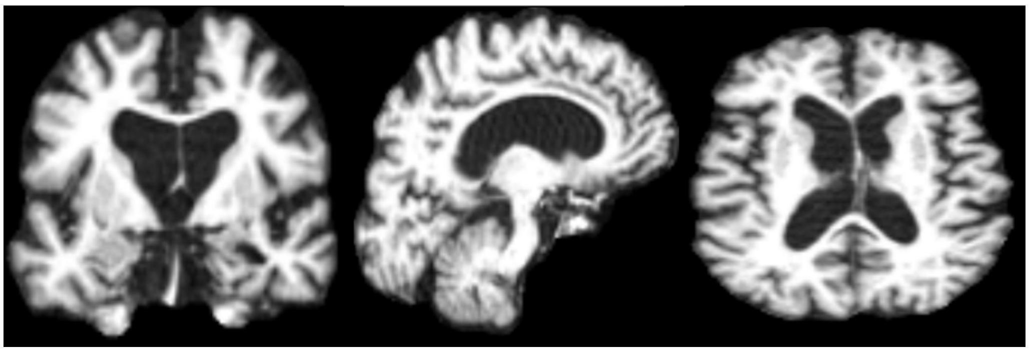


Figure 1.2: Examples of slides from a 3D MRI scan of a patient diagnosed with Alzheimer's disease (from dataset D_1, see section 4.1.1.2).

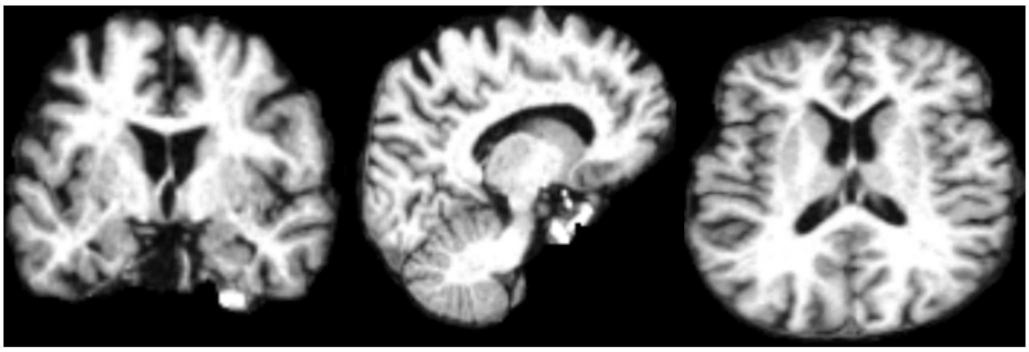
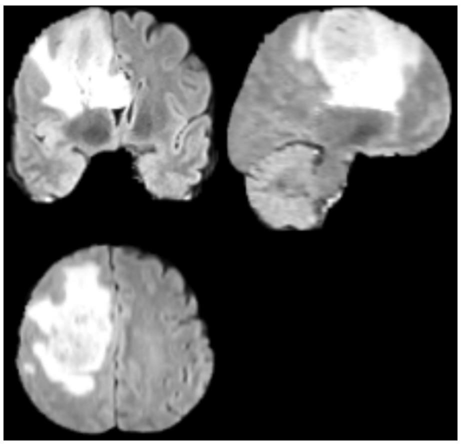


Figure 1.3: Examples of slides from a 3D MRI scan of an elderly healthy person (from dataset D_1, see section 4.1.1.2).

1.2 Brain Tumor

A glioma is a type of malignant **Brain Tumor** (BT) that origins in the brain or spine. The symptoms of a glioma depends on it's position, but can include headaches, vomiting, seizures, or even visual loss if the optic nerve is affected. The exact causes of gliomas is yet unknown, although some risk factors have been investigated, such as **age**, **radiation exposure**, **genetics**, **diet** with varying results [5]. There are several glioma grading systems, one of the most common one is the World Health Organization (WHO) [6]. Gliomas can be divided into **low-grade** gliomas (WHO grade II) and **high-grade** gliomas (WHO grade III-IV). Low-grade gliomas tend to exhibit *benign* tendencies, and hence a better prognosis for the patient, but increase in grade over time, and is hence classified as malignant. The prognosis for patients suffering from a glioma is generally poor. It has been reported that for low-grade gliomas, the age-standardized 10-year relative survival rate was 47% [7]. However, for high-grade gliomas the median overall survival for WHO grade III is approximately 3 years, and for glioblastoma multiforme a poor overall survival of around 15 months [8].


(a) HGG

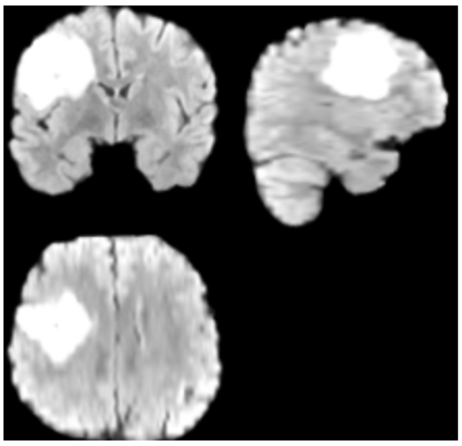

(b) LGG

Figure 1.4: Slices of FLAIR MRI 3D images corresponding to both low and high grade gliomas

Figure 1.5: WHO glioma grading system

Type	Grade	Description	Median survival (years)
Astrocytoma	II	Found diffusely infiltrating into surrounding neural tissue; increased hypercellularity, no mitosis	6-8
Oligodendroglioma	II	Occur in the white matter and cortex of the cerebral hemispheres, low mitotic activity, no necrosis	12
Oligoastrocytoma	II	Diffuse mixed tumor with mixed glial background	3 to >10
Anaplastic-astrocytoma/ oligodendroglioma	III	Highly infiltrating tumors with increased mitotic activity; no necrosis or vascular proliferation	3
Glioblastoma	IV	Infiltrating glial neoplasm with necrosis and micro-vascular proliferation; high rate of mitosis	1 to 2

1.3 Machine Learning

One machine learning topic is **Artificial Neural Networks** (ANNs) (see full definition in Theory), which is a computational system that has been proven useful in problems such as classification, pattern recognition, and regression. An ANN consists of a set of *neurons* with a corresponding *weights* that controls the influence of the signals between them (see details under Theory). The training process of an ANN consists of adjusting the *trainable parameters* of the network, e.g. the above mentioned neuron weights, using an iterative optimization method based on **Stochastic Gradient Descent** (SGD). This gives rise to the problem of efficiently computing the gradients of the performance of the ANN, for which, in 1986, Rumelhart, Hinton and Williams [9] proposed the **back propagation** algorithm to solve, and showed experimentally it's usefulness in practice. Back propagation was quickly adopted by the front line researchers of machine learning at the time. An example of this is LeCun who successfully performed handwritten digit recognition using artificial neural networks, trained with back propagation [10], in combination with feature maps and weight sharing techniques. Later on, this structure became known as **Convolutional Neural Networks** (CNN) [11]. During this time, many advances using ANN variants were made, an example of which is the **Long Short-Term Memory** (LSTM) unit [12] proposed by Hochreiter et al. in 1997. The LSTM unit is a type of stateful neuron to be used in **Recurrent Neural Networks** (RNN), which solves the problem of long-term dependencies. The long-term dependency problem was investigated more in detail 1994 by Bengio et al. in [13]. These are examples of several advances that were made during this time using ANN:s, but still, they were outperformed by analytical machine learning methods such as **Support Vector Machines** (SVM) [14] and **Gaussian processes** [15]. One reason for this can be that the training process of ANN is computationally expensive, and the running time scales rapidly in the number of trainable parameters. As computer power became more easily accessible, ANN approaches grew more common. Even though the training process of an ANN is in general more computational heavy, the structure is more flexible than e.g. SVM with regards to the input, and also often faster in the *prediction step*. Ever since, the research on CNN:s started growing rapidly, and proposed structures such as AlexNet [16], VGG [17], GoogLeNet [18] have been found to perform excellently, even super-human, on tasks such as character/digit recognition, traffic sign recognition, image classification and similar problems.

1.4 Convolutional Neural Networks and Deep Learning

Convolutional neural networks [11] consist of convolutional layers, that makes use of the efficiently computable gradients to allow a number of *filters* to be trained to recognize *features* in an image. The layer outputs a feature map that is the result of a convolution of each filter with the pixels of the image. Since each filter is applied at several positions in the image, **weight sharing** is achieved which results in both a greatly reduced number of trainable parameters, as well as translation-invariant feature detection. It is common to use several consecutive convolutional layers to form **deep convolutional neural networks**, where the resulting feature map from one layer is input to the next. This allows the system to learn a hierarchical abstract representation of features from complex data, a process known as **deep learning**. As mentioned above, this way of learning has been empirically proven to be advantageous for replacing humans in classification tasks. The reason for this might be that it is similar to the way the human sensory systems work; some studies have shown that the human visual perception, as well as speech production systems processes the data in layered hierarchical structures [19].

1.5 Aims of this project

In this project, we aim to address mainly the following issues:

- Deep learning methods, more specifically 3D CNNs, for Alzheimer's disease detection and brain tumor classifcation, using MR images.
- Finding the optimal choice of CNN architecture and hyperparameters of the training process.

2
Background Theory and Methods: review

In this chapter, we review the theoretical background of the methods that are related to our study. We begin by discussing other works that are related to ours. Then, to further increase the understanding of the underlying theory of our methods, some fundamental concepts of machine learning with artificial neural networks are introduced.

2.1 Overview of Related Works on AD Detection

In this section, we give an overview of publications related to our thesis, and compare their results with ours.

DeepAD: Alzheimer's Disease Classification via Deep Convolutional Neural Networks using MRI and fMRI [20]
In this article, pipelines for processing both Magnetic Resonance Imaging (MRI) and functional MRI (fMRI) images are described, and a *deep convolutional neural network* model is trained to classify these images into either Alzheimer's Disease (AD) of Normal Control (NC). Two different sets of data were used for training and testing the model, which were of sizes 144 (AD: 52, NC: 92) and 302 (AD: 211, NC: 91). Methods of augmenting the data is then used in order to generate an incredible amount of several hundred thousands of input images, with which the model is trained. In this work, they combine the output probability distribution of the CNN with what they call a *decision making algorithm*, which chooses to disregard the output of the CNN in ambiguous cases, i.e. if the probability of one class is not sufficiently higher then the probability of the other. This is a relevant step towards developing a system that is applicable in a clinical setting, since if the output of the system does not show certainty in it's prediction, it should not be taken into account in the diagnostic process. The threshold of allowance for uncertainty can then be tuned, to allow arbitrarily high performance, which might be why several cases studies in this article presents a resulting accuracy of 100%. The details of this decision-making algorithm, and how it's parameters were tuned are left out of this article. However, for our purpose, it is relevant only to compare the performance of the CNN component of their system, since this is the main topic of our thesis. The best performance in this related work was achieved using the GoogLeNet architecture, with which they achieved a classification accuracy of 98.84%. The class

distribution of the datasets used, as well as a confusion matrix that shows the class specific sensitivities of their model, are also left out of this article which makes comparison with our results more difficult. Of the work that has been done on this topic, this article represents the current **state of art**. In the Conclusion of this thesis, we compare the methods and corresponding results in this article with the the ones used in this thesis, which is a simple and straight-forward CNN structure, keeping the original structure of the input MRI image intact and analyzing it using 3-dimensional convolutions. Also, our results are achieved with a larger dataset and a simpler preprocessing pipeline. In this thesis, we also make an effort to declare all details regarding the performance, such as the class distribution, class sensitivity and specificity, and discussing correlation between training and test datasets due to several images per patient.

Predicting Alzheimer's disease: a neuroimaging study with 3D convolutional neural networks [21]
This project makes use of 3D convolutional neural networks, trained to perform classification into three classes: AD, MCI (mild cognitive impairment) and NC, and the resulting accuracy in AD-NC classification is 95.39%. The dataset used is claimed to be downloaded from ADNI, and to consist of 755 patients in each class, even though it is specified on the fact sheet from ADNI [22] that their projects in total has included only 350 patients diagnosed with AD. In this paper it is mentioned that their data was provided from [23] in which the number of NC/AD subjects were 232/200. This type of confusion regarding the dataset used for training and evaluation makes comparison more difficult.

Alzheimer's disease diagnostics by adaptation of 3D convolutional network [24]
Dataset consists of 70 diagnosed AD patients and 70 NC. Accuracy for NC/AD classification is 97.6%. The model used in this work combines a *pretrained* model for feature extraction with additional task-specific layers.

Earl Diagnose Of Alzheimer's Disease With Deep Learning [25]
This work makes use of stacked, sparse auto encoders, in combination with a softmax regression layer. Their dataset consists of 65 patients diagnosed with AD, and 77 NC. Highest classification accuracy achieved is 88.57%.

2.2 Machine Learning and Deep Learning

The fundamentals of deep learning begin with artificial neural networks, which will be introduced in the next section. Note that in this project, **supervised learning** is used, which means that we apply machine learning methods using labeled training data.

2.2.1 Artificial Neural Networks

An Artificial Neural Network is a computational structure that consists of one or more computational units known as neurons (defined below). A neuron acts as a mathematical function $\mathbb{R}^n \rightarrow \mathbb{R}$, where the elements of the input vector are the outputs of other neurons (nodes) and the output later on an input to a different neuron, with the exception of so called **input** and **output** neurons. The input to the **input neurons** of an ANN are given by the user of the ANN, usually a computer program, who also fetches the output of the **output neurons** once the computation of an ANN is finished. The normal usage of an ANN is hence that the input neurons are fed some data by the user, and after that data has been processed and passed through the neurons of the ANN, the output of the output neurons is collected. It is desirable if the output given by the output nodes is some non-trivial, more easily interpreted information about the input data, for example the **class** of the input in the case of classification problems. The design of more Complex ANN designs that consist of *more than one* hidden layer are commonly refereed to as **Deep Neural Networks** (DNN).

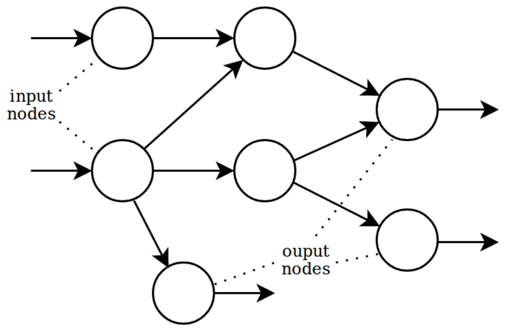


Figure 2.1: Simple artificial neural network

2.2.2 Neurons

A neuron is, as mentioned above, a function $\mathbb{R}^n \rightarrow \mathbb{R}$ where n is the number of inputs to the neuron. Each neuron has a set of internal parameters, namely a weight vector $\mathbf{w}$ and a bias b. The output y of a neuron, given inputs $\mathbf{x} = (x_0, \ldots, x_n)$ is computed as follows

$$y = f(\mathbf{wx} + b) \tag{2.1}$$

where φ is the **activation function**, the definition of which follows in the next section. An ANN with a set of neurons for a fixed set of internal parameters (weights, bias) is known as a *model*.

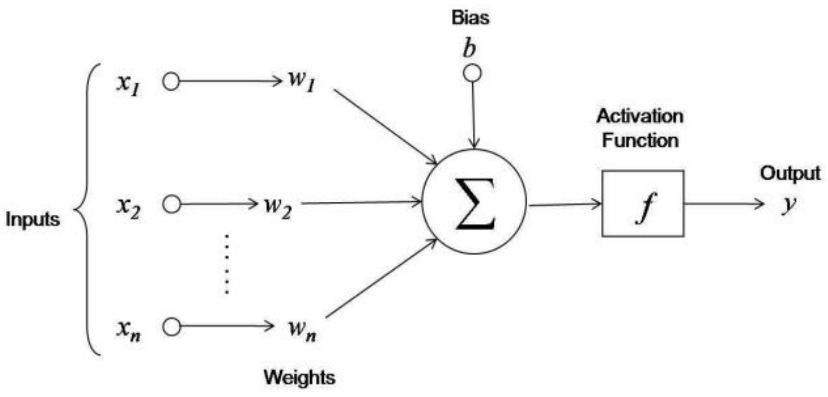


Figure 2.2: Visualization of a **neuron** [26]

2.2.3 Activation Functions, Nonlinearity

The choice of **activation function** for the neurons of an ANN is vital for the network's ability to process nonlinear input data. The activation function should be chosen in a way such that **nonlinearity** is introduced to the system. Otherwise, the ANN will merely be an affine transform of the input to the output, since it is a composition of such transforms. The ANN would hence not be able to, for instance, be useful when it comes to regression problems with non-affine functions. The introduction of a nonlinear activation as in eq. (2.1) solves this problem, and the non-linearity that is introduced can then be scaled, and shifted, by the **weights** and the **bias**. The activation must also be differentiable, the reason for which is shown in the section **Back Propagation Algorithm** below.

Below are some common choices of activation functions.

Hyperbolic tangent

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1 \tag{2.2}$$

Arcus tangent

$$f(x) = \arctan(x) \tag{2.3}$$

Rectified linear unit (ReLU)

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x >= 0 \end{cases} \qquad f'(0) = 0 \tag{2.4}$$

2.2.4 Feedforward Neural Network

Considering an ANN as a directed graph, we define a **Feedforward Neural Network** (FNN) as an ANN that is acyclic. This means that during the computation of one input to an ANN, the output from any node will never, through other nodes or directly, reach that node again. This is not the case for a general ANN, and if an ANN is not an FNN it is called a **Recurrent Neural Network** (RNN). For the remainder of this text, when ANN:s are mentioned, it should be interpreted as an FNN, unless it is explicitly denoted RNN.

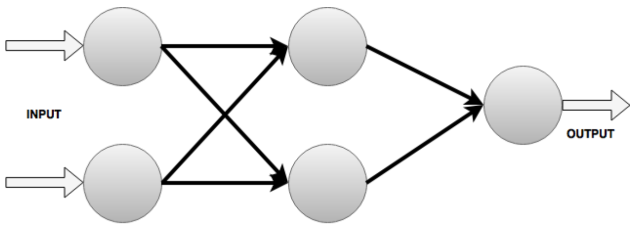


Figure 2.3: Visualization of an FNN
Feed-forward neural networks are usually considered organized into layers, with an *input* and *output* layer. The remaining layers, i.e. the ones between the input and output layers, are known as hidden layers. For an FNN, the data will be passed monotonously through the layers in one direction, and never go backwards.

2.2.5 Classification Problems

A common way to use ANN:s in classification problems is to use *categorical* output, which means that the output layer of the ANN will have the same number, N, of nodes as classes, each node representing one unique class. It desirable that the ANN outputs a probability distribution of the class of the input, i.e. that each output node outputs the probability of the input belonging to the node's corresponding class. This is achieved using the special activation function **softmax**

$$f(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{i=1}^{N} e^{z_i}} \quad \text{for } j = 1, \ldots, N \tag{2.5}$$

where z_j is the output of output neuron j before activation ($z_j = \mathbf{W}_j\mathbf{x}+\mathbf{b}_j$) and $f(\mathbf{z})_j$ is the final output of node j. The class of the input is now naturally estimated as the one corresponding to the output node with the largest output. Note that unlike the activation functions above (Eq. 2.2-2.4), the input to the softmax function is based on all the nodes of the output layer, instead of just one.

2.3 Deep Learning

Complex ANN designs that consist of *more than one* hidden layers are commonly referred to as **deep neural networks**, the training process of which is known as **deep learning**. There is however no universally agreed upon threshold for the number of hidden layers required for the ANN to be referred to as *deep*. DNN:s allow composition of several nonlinear transformations throughout the layers, and hence enable higher level feature learning by 'late' layers. This is based on low level features learned by preceding layers, allowing in the end for a more sparse hierarchical representation of the features. DNN:s has been been proven useful in applications like feature extraction and pattern recognition in combination with convolutional layers.

2.3.1 Convolutional Neural Networks

When a neural network consists of one or more layers known as **convolutional layers**, the ANN is referred to as a convolutional neural network (CNN). The convolutional layer is a layer specifically designed for translation-invariant feature extraction, i.e. features independent of position in the input data. The layer has a fixed number of *filters* (also known as *kernels*), typically with small receptive fields, the parameters of which are optimized during the learning process. In a convolutional layer, each filter is convolved across input, computing the dot product of the filter coefficients and the input, giving a so-called *activation map* of that filter. This gives convolutional layers the ability to learn filters that activate when detecting a specific feature at any position in the input. The list of activation maps from all filters of one convolutional layer is the the output for the layer.

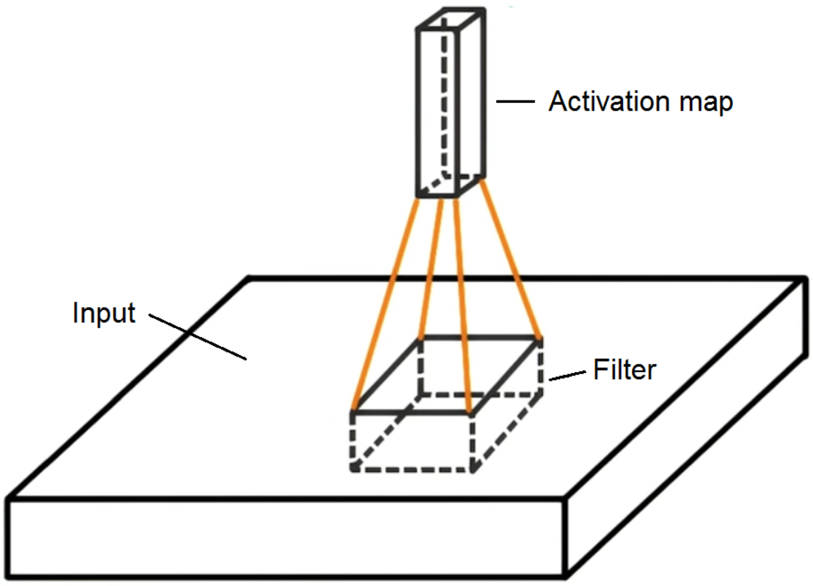


Figure 2.4: CNN Visualization

The convolutional layer has three fixed parameters that are not learn-able, also known as *hyperparameters*

- **Number of filters** i.e. how many filters to be convolved over the entire input, which represents how many different types of features to be extracted in the layer.
- **Stride** is the step length for the convolution. When the stride is 1 the filter is moved one pixel at a time, for stride 2, two pixels at a time, etc.
- **Zero-padding** i.e. how the problem of changed spatial dimension of the output is to be handled. For some cases, it can for instance be appropriate to pad the output so that the spatial dimension of the input is maintained.

The spatial dimension of the output O is easily computed given the input dimension

I, the filter/kernel size K, the stride S, and the zero-padding P used to fill the edges

$$O = \frac{I - K + 2P}{S} + 1 \tag{2.6}$$

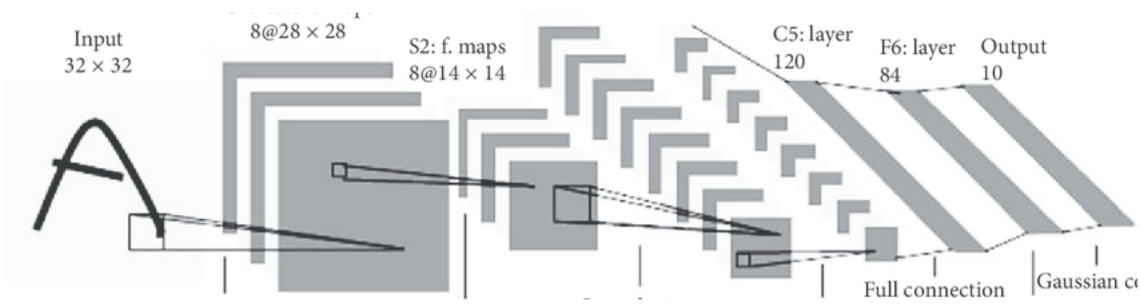
where the division is component-wise for multi-dimensional input. Naturally, K, P, and S are chosen so that O is an integer.

An efficient way to reduce the dimensionality of the data between layers is by using **spatial pooling**, which performs a non-linear down sampling. The most common non-linear pooling method is *max pooling* which divides the data into disjunct rectangles, for each the maximum value is output. This can be justified by arguing that the exact position of the feature-matching data is not as important as the fact that the maximum of the activations within a neighborhood of that data indicates weather there is a match in that region at all. For spatial pooling, the **kernel size** is a hyper-parameter i.e. the size of the rectangle within which to output the maximum, for which a common choice is kernel size 2.

Below, we show some successful well-known CNN architectures:

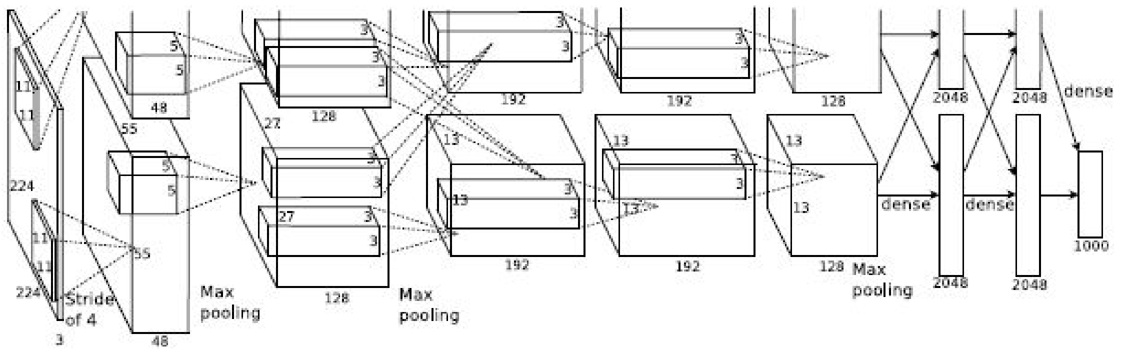
LeNet, 1998 [27] LeCun et al.
As mentioned in the introduction, LeCun was the first to make use of *weight sharing* and *feature maps* in neural computing, a technique which later on would be known as a convolutional layer. In 1998, LeCun proposed a CNN architecture known as LeNet (Figure 2.5), that performed well on document recognition, which is to say handwriting recognition.

Figure 2.5: LeNet-5 architecture, by LeCun et al. 1998 [27]



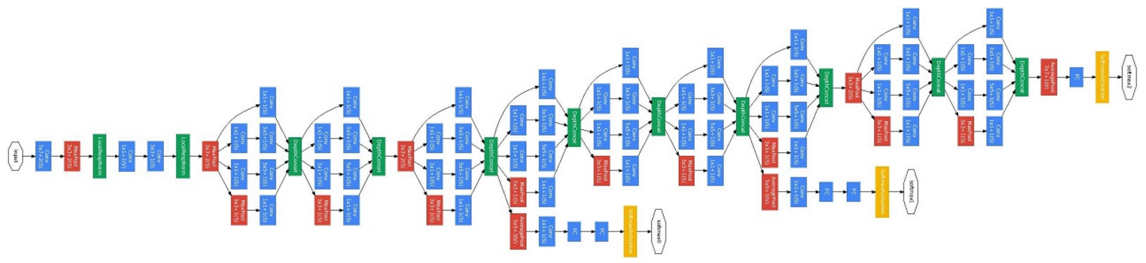
AlexNet, 2012 [16] Krizhevsky et al.
AlexNet is a deep convolutional neural network, shown in Figure 2.7, that was the winner of the ImageNet ILSVRC challenge in 2012 [16]. Is is named after one of its designers, Alex Krizhevsky, and consists of 650,000 neurons, and 60 million parameters. The structure is *deeper* and *wider* than LeNet-5, and is unique in the way it makes use of 11 × 11 receptive fields for filters, that were convolved with stride four. The structure is tailored to the way the system is implemented, namely using two GPU:s in parallel, that both execute one stream of the network each, with limited communication in between.

Figure 2.6: AlexNet architecture, by Krizhevsky et al. 2012 [16]



GoogLeNet, 2014 [18] Szegedy et al.
In 2014, Szegedy proposed a deep convolutional network architecture known as **inception** [18], the idea of which is to use an optimal sparse structure, that consists of several dense components. This enables a very deep structure, with a lower number of parameters. GoogLeNet was the winner of the ILSVRC challenge in 2014.

Figure 2.7: GoogLeNet architecture, by Szegedy et al. 2014 [18]



2.3.2 Recurrent Neural Network

In terms of our definition of an ANN above, a **recurrent neural network** (RNN) is, as mentioned, a cyclic ANN. This means that information can propagate in cycles through the network, which can be used to allow the ANN to have an internal memory to store information regarding previous inputs. In most applications, these cycles consists of only one neuron (sometimes called **recurrent unit**), see Figure 2.8. This way, omitting the bias, the output of a neuron at time t is slightly changed to

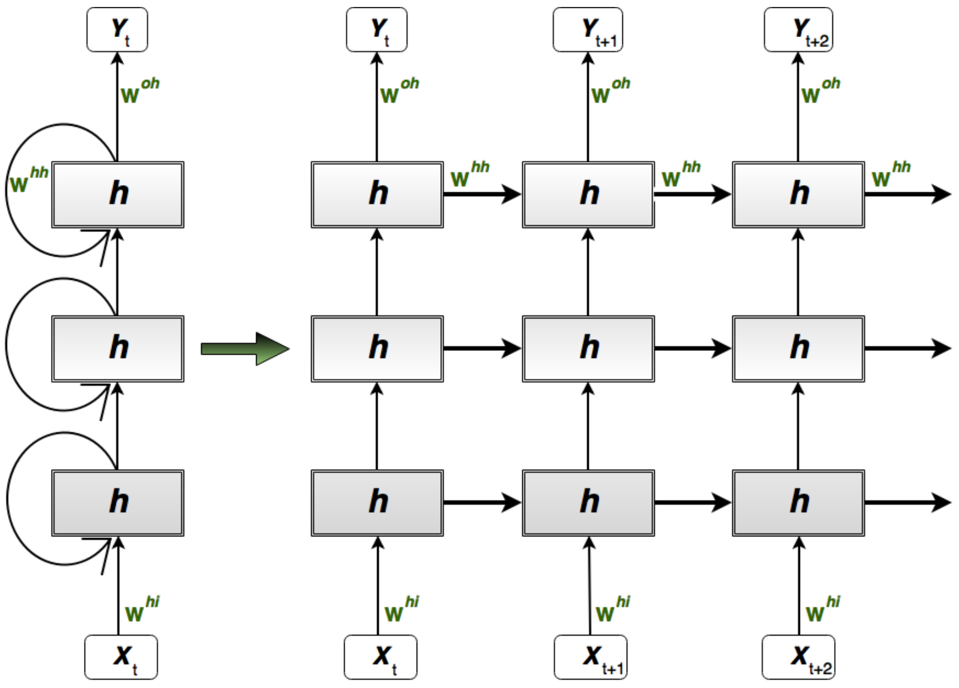
$$y_t = f(W^{oh} s_t) \tag{2.7}$$

where f is an activation function, and s_t is the state of the neuron at time t, defined as

$$s_t = g(W^{hi} x_t + W^{hh} s_{t-1}) \tag{2.8}$$

where x_t is the input to the neuron at time t, g is an activation function, $H^{(hi)}$ and $H^{(hh)}$ are weights associated with the neuron. The neuron weights are trainable using back propagation, since eqs. (2.7) and (2.8) both constitute differentiable transformations.

Figure 2.8: RNN with three layers, unrolled over time



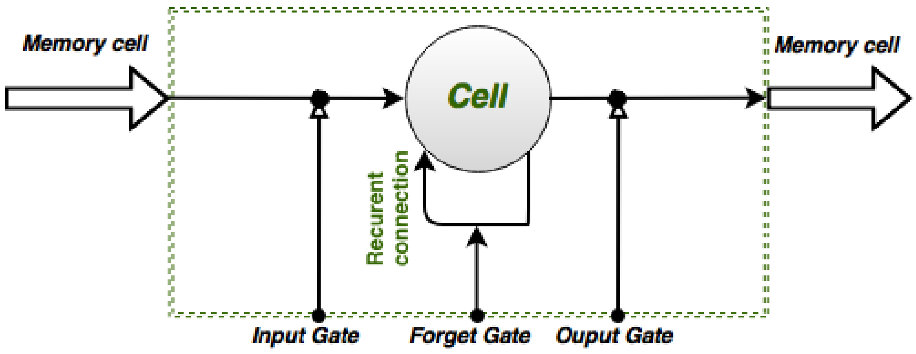
2.3.2.1 Long Short-Term Memory (LSTM)

RNN:s are promising for solving problems that require consideration of previous inputs, i.e. what has happened before, e.g. predicting an event in a video, which would require at least considering the last few frames of the video. However, for applications that are more dependent on *temporal* information, i.e. require consideration of input that could have been given at a much earlier time, general RNN:s fail to achieve convincing performance. This problem is known as the **long-term dependency** problem, and is apparent in applications such as translation, and other time-dependent event prediction. The problem arises in the back propagation algorithm, when generalized to take the temporal dependency into account. The reason for this is that $H^{(hh)}$ is applied to the state arbitrarily many times, or more precisely as many times as there are time steps. That way, during one pass of back propagation, if $H^{(hh)}$ is small, or more precisely has an *eigenvalue* λ such that $\lambda \in (-1, 1)$, the gradient signal over time will vanish as the number of time steps increase. Similarly, if $\lambda \notin (-1, 1)$, the signal can grow out of proportion, and be rendered useless. These phenomena are known as the **vanishing gradient problem** and **exploding gradient problem**, respectively.

The LSTM[12] unit solves the above mentioned problems, by introducing a specific structure for storing information, known as the **memory cell** (see Figure 2.9). The memory cell consists of four parts: an input, output and forget gate, and a neuron.

The vanishing/exploding gradient problem is avoided by letting the re-currency weight be identity, which means that the state of the memory cell is not scaled over time. The influence of the input over the memory is determined by the input gate, which can either block the signal, or allow it to alter the state, and the output gate can decide to block the output of a cell, or allow it to influence other neurons.

Figure 2.9: Basic LSTM structure



2.4 Training the Network

In this section, we introduce the concept of *training* an ANN for a specific task. We begin with defining some ways to measure the *performance* of an ANN.

2.4.1 Performance Measures

The performance of a model (recall, an ANN for a fixed set of weights and biases) is a measure on how well it performs its computational task on certain input data, which is measured by comparing the model's output with the *desired* or *correct* output. In the example of a classification problem, the performance of a model is good if the model output is often corresponding to the true class of the input.

The concept of *training* an ANN is the process of altering the *trainable parameters* of the ANN, namely the weights and biases of the neurons, so that the resulting model has a higher performance. Training an ANN is hence an optimization problem, namely optimizing the performance in the parameter space.
The performance is usually measured in two ways, **Loss** and **Accuracy**

2.4.1.1 Loss

The **Loss** is a measure on how large the error of the model is, and can be defined in different ways. Below we descrobe some common ways to define loss.

- **Mean-squared error**

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i^{\text{pred}} - Y_i^{\text{true}})^2 \tag{2.9}$$

where n is the input dimension, Y_i^{pred} is the model output for the i:th input, and Y_i^{true} is the correct output.

- **Categorical cross-entropy**

$$H(p,q) = -\sum_x p(x) \log q(x) \tag{2.10}$$

where p is the *true* probability for x, and q is the model output.

2.4.1.2 Accuracy

Accuracy over a dataset is the percentage of *correct outputs*

2.4.2 Parameter Optimization

The *training* of an ANN can now be defined as an optimization problem, namely to minimize the loss over the space of trainable parameters. The most common approach to this problem is to use optimization algorithms based on **stochastic gradient descent**[28], i.e. in each iteration moving in the direction of the steepest descent in parameter space, in turn lowering the loss. This approach requires that the gradient of the loss function can be efficiently computed, which is accomplished by using the **back propagation algorithm** (see below). There are many different specialized algorithms for ANN training, that are all based on SGD, e.g. Adagrad[29], Adam[30], Adadelta[31] and others.

2.4.3 Back Propagation Algorithm

We introduce the idea of the Back Propagation algorithm (BackProp) taking only the neuron weights into account, and omitting the bias. For a full description see [32]. The principle of the BackProp algorithm is to calculate the gradient of the loss function with respect to each trainable parameters of the ANN, i.e.

$$\frac{\partial L}{\partial w_{ij}^k} \tag{2.11}$$

where w_{ij}^k is the i:th weight of node j in layer k, which will allow the optimization algorithm to update the weights step by step using SGD

$$w_{ij}^k = w_{ij}^k - \eta \frac{\partial L}{\partial_{ij}^k} \tag{2.12}$$

where η is the step length, in this context known as the **learning rate**. Note that η can be varied during training, depending on if optimizer is *adaptive* or not.

The gradient computation is achieved as follows

1. The first step is to do a *forward pass* of the the ANN, i.e. feeding data to the input layer, and computing the loss of output. In the output layer, given the

activation and loss function, the loss gradient with respect to each weight of the neurons in the output layer can be computed.

$$\frac{\partial L}{\partial w_{ij}^{o}} \tag{2.13}$$

where w_{ij}^{o} is the i:th weight of node j in the output layer.

2. Using the chain rule, one can, using (2.13), calculate the gradient of the loss with respect to any weight in the last hidden layer as follows

$$\frac{\partial L}{\partial w_{ij}^{o-1}} = \frac{\partial y_i^o}{\partial w_{ij}^{o-1}} \frac{\partial L}{\partial y_{ij}^{o}} \tag{2.14}$$

where y_i^l is the output of neuron i in layer l. Here it is made use out of that the *local* gradients of the neurons can be easily computed, and used for computing the gradients for the previous layers. Similarly to (2.14), all gradients are computed recursively, and finally the complete loss gradient with respect to all ANN parameters can be computed, and the SGD algorithm iterated. This step is known as the *backward* pass.
Using this, a *backward pass* is done, where we use the chain rule to, layer by layer, calculate the loss gradient with respect to the parameters of each neuron.

2.4.4 Training Data

The set of inputs that are fed to an ANN during the training process is called the **training data**. The size of the training data, and the structure of the individual inputs play an important role when deciding on the ANN structure, for instance when it comes to image-like inputs, using convolutional and spatial pooling layers in the first layers of the network will help extract important features from the image, and reduce the dimensionality for following layers, without the need to deal with the optimization of a huge amount of parameters unnecessarily.

2.4.4.1 Test Data

A common way to determine the general performance of a model is to have a separate set of data, known as the **test data**, on which the model is tested, and the resulting *test loss* and *test accuracy* is noted. During the training process of an ANN, no information (such as size, dimensionality, input examples etc) regarding test data is used, since if it was, the test performance is less likely to reflect the general performance of the model.

2.4.5 Overfitting

A problem within machine learning is what is known as **overfitting**, which means that a system adapts to *noise* in the data, where by noise we mean information that is uncorrelated to the features we wish our machine to learn.

2.4.5.1 Regression Problems

A simple example of overfitting in regression is the problem of fitting a polynomial to a set of data points $\{\mathbf{x}_i\}_1^{N+1}$, namely modeling the data by a N-degree polynomial. This is sufficient for the approximation error to vanish for the right polynomial coefficients, but is most likely not a useful model when applied for more data from the same distribution.

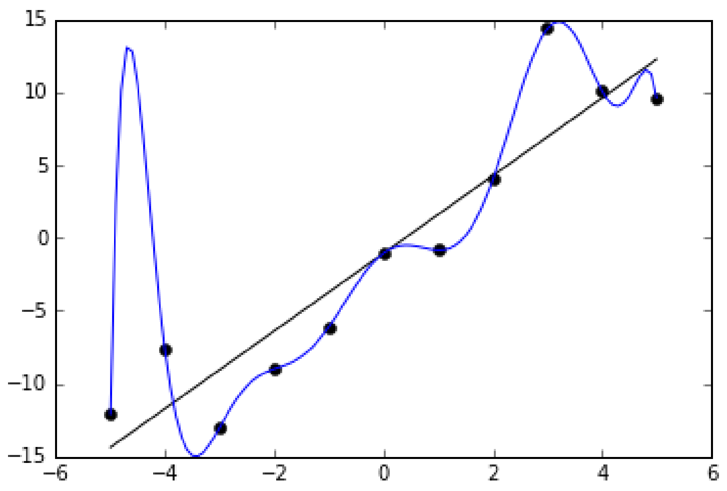


Figure 2.10: Overfitting in regression
Data approximated by a linear function and a higher-degree polynomial. The polynomial is a perfect fit, but might not be expected to perform well in general. (By Ghiles - CC BY-SA 4.0)

2.4.5.2 Overfitting in an ANN Classifier

When it comes to training of ANN classifiers, one can expect that it will result in a model that has learned the simplest way to sort the inputs into different classes. An example of overfitting here would be if for instance we wish to train an ANN to classify pictures into to classes, *dogs* and *cats*. If all the pictures of dogs happen to have green grass in the background, while that is less common in the pictures of cats, the training might lead to a model that only takes the color of a few pixels in one corner into account, and based on that performs the classification. A way to prohibit this would naturally be to have more, and more diverse training data, where there are pictures of both dogs and cats with several different background environments, which hopefully will enforce the training process to yield a model that has learned more relevant features.

2.4.5.3 Prevention

As mentioned, adding more data can help the classifier to focus on the desired features. Other methods to prevent overfitting includes:

- **Data augmentation**, i.e. increasing the amount of training data by altering it by some transformation. E.g. in the example of pictures of dogs and cats, a simple data augmentation would be to simple flip all pictures horizontally, which would result in a training data set of twice the size. Depending on the features that the classifier should learn, other augmentation methods could be used for images, such as warping, color or intensity altering.

- **Batch normalization**, normalizes the input data to layers in the ANN, and thereby reduces the *internal co-variance shift.* Batch normalization has not only been found to reduce overfitting, but also speed up the training by allowing higher learning rates[33]. The effect of normalization is decided by *trainable parameters*, and thus the model can *learn* to deactivate the effect.
- **A generalizing architecture**, which means that the ANN structure should be chosen so that it has the parameters necessary to learn the desired features, and not too many more.
- **Use regularization** (defined below)

2.4.6 Regularization

In this section, we describe some methods for counteracting overfitting in an ANN classifier.

2.4.6.1 Early Stopping

We introduce a new term **Validation data**, by which we mean a subset of the training data, the elements of which are *not* used in the training process. The validation data is instead used to occasionally *validate* the performance of the model. Measuring the performance on the validation data will give an indication on how the general performance of the model is, and overfitting is easily detected by noticing that the *training accuracy* (performance accuracy over the training data) continues to increase, whilst the performance on the validation data worsens. This is caused by that the classifier has learned undesirable features that are sufficient for performing well on the training data, but not in general. In this scenario, it is unlikely that the training process will recover, and that the validation performance will increase eventually (since this would the require the model to 'unlearn' the currently learned features, before learning the desirable ones), and it is hence usually advantageous to stop and reinitialize the training process. This is known as **Early stopping**, and instead of using the model finally output by the training process, the model that during the training had the *highest validation performance* is considered the result of the training, and is then tested on the *test data.*

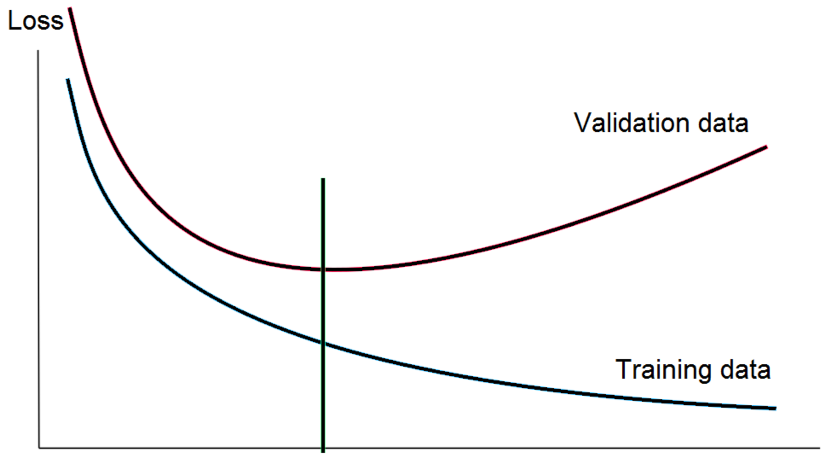


Figure 2.11: Early stopping

2.4.6.2 L-Regularization

Also known as **weight decay**, this regularization method consists of penalizing the loss function by adding a multiple of the norm of the model weights. Two common norms used for weight decay are

$$L_1 \text{ norm:} \qquad \lambda \sum_{i=1}^{k} |w_i|$$

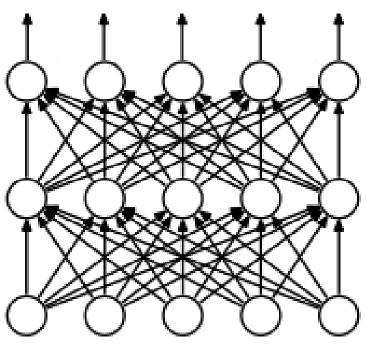
and

$$L_2 \text{ norm:} \qquad \lambda \sum_{i=1}^{k} w_i^2$$

Weight decay promotes *weight sparsity* and has been found to improve the generality of the model and reduce overfitting [34].

2.4.6.3 Dropout

A simple, but effective method for reducing overfitting, known as **dropout**[35], forces the model to learn general features by *dropping out* neurons in the computation, i.e. ignoring the output of these neurons. This prohibits so-called *specialized* neurons that can allow for the ANN to memorize the training data, rather than learning the desired general features. The neurons to drop out are selected at random, with a certain fixed probability denoted by p_d, given as a hyperparameter.


(a) Without dropout

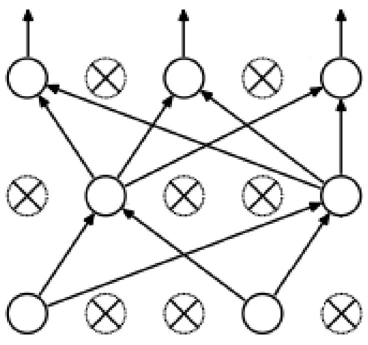

(b) With dropout

Figure 2.12: Dropout visualization

2.5 Weight Initialization

In this section, we describe an important step in the training process that can be vital in succeeding in training a useful model, but often overlooked (or unmentioned) by researchers in the field. The gradient descent-based optimization method of the parameters naturally requires an **initialization** in parameter space, which can be achieved in many ways. One easy way is to initialize all parameters to zero, with the hope that a global optimum is close to, or easily found from, origo. There is however no justification for this, and this initialization method has accordingly not been found to be useful in practice. Though SGD is used, the training process of ANN:s in practice often ends up in local optima. That is why, it is a good idea to run the training several times, and vary the weight initialization. To this end, the most common parameter initialization is random, from a standard distribution with low variance.

2.5.1 Xavier Initialization

For convolutional layers specifically, it is common to see Xavier initialization being used, which helps initialize the weights within a reasonable interval so that the signals passing through the layers of a deep network are not scaled several times by layer weights too large/small, so that they eventually could be rendered useless. This is achieved by generating the weights from a distribution so that the variance of the output of a neuron will be same as the input, which if we for simplicity consider a linear neuron, gives

$$\operatorname{var}(y) = \operatorname{var}(w_1x_1 + w_2x_2 + \cdots + w_nx_n + b) \tag{2.15}$$

$$= \sum_{i=1}^{n} \operatorname{var}(w_ix_i) + \operatorname{var}(b) \tag{2.16}$$

$$= \sum_{i=1}^{n} \operatorname{var}(w_ix_i) \tag{2.17}$$

from linearity of variance, where b is a constant. We have, assuming that w_i and x_i are independent

$$\mathrm{var}(w_i x_i) = \mathrm{E}(w_i)^2 \mathrm{var}(x_i) + \mathrm{var}(w_i)\mathrm{var}(x_i) + \mathrm{E}(x_i)^2 \mathrm{var}(w_i) \tag{2.18}$$

which, if we assume that w_i, x_i are Gaussian distributed with $\mu = 0$ gives

$$\mathrm{var}(w_i x_i) = \mathrm{var}(w_i)\mathrm{var}(x_i) \tag{2.19}$$

which yields

$$\mathrm{var}(y) = \sum_{i=1}^{n} \mathrm{var}(w_i)\mathrm{var}(x_i) \tag{2.20}$$

which, if we assume identically distributed variables, gives

$$\mathrm{var}(y) = n \cdot \mathrm{var}(w_i)\mathrm{var}(x_i) \tag{2.21}$$

from which it is obvious that y being distributed identically to x implies

$$n \cdot \mathrm{var}(w_i) = 1 \tag{2.22}$$

$$\Rightarrow \mathrm{var}(w_i) = \frac{1}{n} \tag{2.23}$$

Hence, the initial weights of a neuron should be generated randomly from a zero-mean Gaussian distribution with variance $\sigma^2 = \frac{1}{n}$, where n is the number of inputs to the neuron.

2.6 Graphic Process Unit

In recent years, Graphics Processing Units (GPUs) have been utilized for deep learning. Even though the GPU is slower than a Central Processing Unit (CPU) when it comes to sequential processing, it greatly outperforms the CPU in parallel processing, due to it's high number of processing cores. A CPU usually has 4 to 6 cores, while a GPU might have hundreds, or even thousands. The drawback of using GPU for computation is that it has specific requirements on hardware and software, in order to be able to distribute the computational tasks among the cores. Providers of such software is mainly the producers of such GPU:s, which is mainly AMD and NVIDIA. For GPU:s produced by NVIDIA, their own application programming interface **CUDA** is a common choice. Other API:s are available, such as Open Computing Language (OpenCL), which is an open source API that can be used for GPU:s produced by both NVIDIA and AMD.

2.7 Deep Learning Software and Libraries

There are many libraries and platforms for machine learning. Some are open source such as Theano, Theano, Lasagne, Keras, Caffe which we will summarize them in this section. There are some others worth mentioning include TensorFlow by Google (in python and C++), Mocha (in Julia), CNTK by Microsoft (Microsoft Cognitive Toolkit Written in C++), Darch (Package in R), Convnet.js (in JavaScript, for learning) and H2O Web API.

Theano

Theano is a library developed at the University of Montreal [36] for mathematical expressions in Python with the goal of facilitating research in deep learning. It is a back-end engine using tensor for handling and manipulating the data and is centered based on the idea of computational graphs. It uses SciPy-NumPy, native libraries such as native C++ codes to convert a computational graph structure into very efficient codes that can be run in the fastest way possible on CPUs or GPUs in C++ or CUDA. Theano is know to be a good tool for implementing Deep Convolutional Network, Stacked Denoising Auto-Encoders and Deep Belief Networks. Keras, Lasangne, and Blocks are libraries built on top of Theano as wrapper APIs.

Torch

Torch[37] is a scientific library and framework for computation and a script language based on the Lua that highly supports machine learning algorithms and prioritize the GPU. Torch is being used within Facebook, Google, Twitter, NYU, IDIAP, Purdue and several research labs. Lua is a scripting language intended for embedded devices. In Torch graphs of neural networks can be built and then be parallelized over CPUs and GPUs in an efficient way. It allows to a deep network to be built in a sequential way consist of many stacks of layers. Torch is recommended primarily platform for research in reinforcement learning. In Torch many of the state of the art machine learning algorithms are implemented

Caffe

Caffe[38] is a Python library developed by Yangqing Jia during his Ph.D in at the Berkeley Vision and Learning Center for supervised computer vision problems written mostly in C++ for deep learning and it uses python as its API. The primary focus of Caffe is CNN and it may be the best library for this purpose. One of the most interesting and benefit aspect of Caffe is the number of pre-trained networks. These networks can be downloaded from the Caffe Model Zoo and used immediately. Even though Caffe is recommended for CNN but it is not as strong for RNN. On the abstraction level, Caffe is higher than Cuda-Convnet but lower than Torch.

Keras

Keras[39] is a high-level and user friendly neural networks API and wrapper in python designed and consist of a sequence or a graph of standalone from fully configurable modules. Keras runs on top of Theano, CNTK and Tensorflow. Keras will not show any of the underlying work and designed to allow for fast and easy prototyping of machine learning algorithms with as little as possible overheads.

Tensorflow

Tensorflow[40] is a an open-source software library for machine learning developed by researchers at Google Brain with Python and C++ API. It included many pre-built functions to facilitate the built of different neural networks. Tensorflow accommodates the computation distribution across different computers, or multiple CPUs and GPUs in a single machine. It facilitate and speed-up experimentations, while remaining fully transparent. TensorFlow is the second generation machine learning system of Google brain and released as open source and its version 1.0.0 was released on February 11, 2017.

A short summary of these libraries can be found in Table 2.1.

Table 2.1: Deep learning libraries summary

Name	Language	Back-end	Comp. Graph	Open Source/ Readable	Recommended For
Theano	Python	C++, CUDA, OpenCL	Yes	Yes/No	CNN,Auto-Encoders,Deep Belief Networks
Torch	Lua	C,CUDA, OpenCL	Yes	Yes/Yes	Reinforcement Learning
Caffe	C++, Python	C,CUDA, OpenCL	Yes	Yes/Yes	CNN
Keras	Python	Theano, CNTK, Tensorflow	Yes	Yes/Yes	-
Tensorflow	Python, C++	C++, CUDA, OpenCL	Yes	Yes/No	RNN

3
Methods

In this chapter, we describe the methods we have developed for brain image analysis. Our work can be divided into two parts, Alzheimer's Disease (AD) detection and Brain Tumor (glioma) classification. Since the methods used for the two parts are similar, they will be described in general below, and any differences will be pointed out.

We aim to develop a system that can perform brain disease detection/classification based on an MRI brain scan of the patient. We propose to do this using a **deep convolutional neural network**, trained and evaluated on datasets consisting of MRI brain scans. The effect of certain aspects of the CNN architecture will be investigated by comparing the performance of different architectures. The importance of some hyperparameters such as *dropout probability*, *learning rate* will also be investigated, by *optimizing* the system performance with respect to these hyperparameters.

Figure 3.1 shows an outline for the classifier training process. Early stopping is used to determine when to terminate training. This means that the performance of the system is monitored on a set independent of the training data, called the **validation data**, and when the validation performance is the highest, the training is terminated, and the resulting classifier is tested on the **test data**.

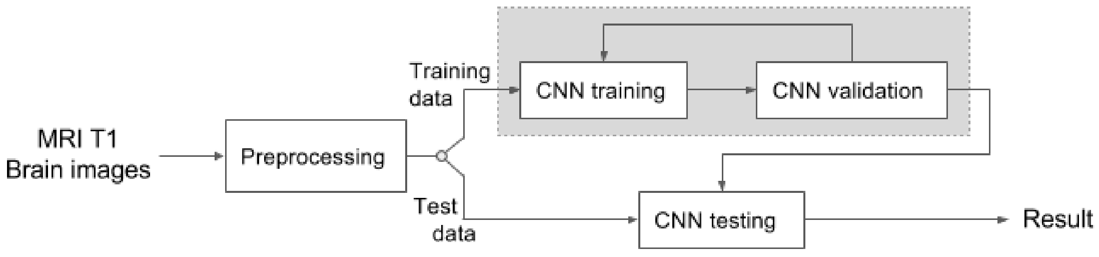


Figure 3.1: Overview of the training process of a CNN

3.1 Selection of CNN Architecture

We propose a **deep convolutional neural network** for the task of brain disease classification from MRI images. The proposed structure has **five** convolutional layers, which is beneficial since this allows a hierarchical representation of the features learned, and is less computationally intensive when used in combination with **pool-**

ing layers. Inspired by AlexNet[16], the first convolutional layer utilizes contains a large kernel size, and stride 2. The remaining convolutional layers use a smaller kernel with stride 1. After the convolutional layers, there are two fully-connected hidden layers that will use the feature activation maps from the last convolutional layer to perform the final classification. The effect of both the choice of kernel and stride in the first layer, as well as the depth chosen will be investigated by comparing the performance of this architecture with **two** others.

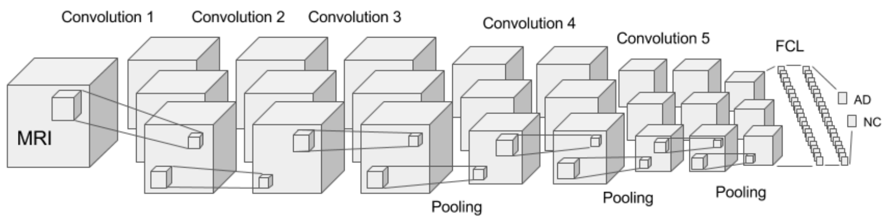


Figure 3.2: 3D-CNN visualization

The details of the proposed CNN architecture are shown in Table 3.1 and is denoted architecture A. We compare architecture A with two other architectures, namely B and C. The details of these architectures are shown in Tables 3.2 and 3.3. More detailed specification of the CNN architectures are included in Appendix A.1, A.2, A.3.

Table 3.1: Architecture A

Layer type	Kernel size	Stride	No.Filters	No.Neurons
Conv1	(7,7,7)	(2,2,2)	64	-
Conv2	(3,3,3)	(1,1,1)	64	-
Conv3 + MaxPool	(3,3,3)	(1,1,1)	128	-
Conv4 + MaxPool	(3,3,3)	(1,1,1)	128	-
Conv5 + MaxPool	(3,3,3)	(1,1,1)	128	-
FC1	-	-	-	256
FC2	-	-	-	256
FC3	-	-	-	2

Table 3.2: Architecture B. Similar to A, except for only the *kernel size* in the first convolutional layer

Layer type	Kernel size	Stride	No.Filters	No.Neurons
Conv1	(3,3,3)	(2,2,2)	64	-
Conv2	(3,3,3)	(1,1,1)	64	-
Conv3 + MaxPool	(3,3,3)	(1,1,1)	128	-
Conv4 + MaxPool	(3,3,3)	(1,1,1)	128	-
Conv5 + MaxPool	(3,3,3)	(1,1,1)	128	-
FC1	-	-	-	256
FC2	-	-	-	256
FC3	-	-	-	2

Architecture *C* is similar to model *A*, except for that one intermediate convolutional layer is left out.

Table 3.3: Architecture C. Similar to *A*, except for that one intermediate convolutional layer is left out

Layer type	Kernel size	Stride	No.Filters	No.Neurons
Conv1	(7,7,7)	(2,2,2)	64	-
Conv2	(3,3,3)	(1,1,1)	64	-
Conv3 + MaxPool	(3,3,3)	(1,1,1)	128	-
Conv4 + MaxPool	(3,3,3)	(1,1,1)	128	-
FC1	-	-	-	256
FC2	-	-	-	256
FC3	-	-	-	2

3.2 Dataset Description

In the following sections, we describe the data for Alzheimer's disease and Brain tumor used in this project, and how it was acquired.

3.2.1 Alzheimer's Disease Detection

Data used in this project was obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database. The ADNI data collection began in 2004 and is now in its third phase from ADNI1 to ADNI GO and ADNI2 [22]. The number of normal control subjects is in total 350, and 400 for patients diagnosed with Alzheimer's disease (see Table 3.4). ADNI provides researchers with study data as they work to determine the progression of AD. ADNI researchers collect, validate and utilize data such as MRI and PET images, genetics, cognitive tests, CSF and blood biomarkers as predictors for the disease. Data from the North American ADNI's study participants, including Alzheimer's disease patients, mild cognitive impairment subjects and elderly controls, are available from this website `http://adni.loni.usc.edu`

Table 3.4: ADNI data and samples [41]

Study Type	No. Subjects (NC/AD)	Type of Images
ADNI 1	400 (200/200)	MRI, FDG, PIB, Biosamples
ADNI GO	500 (500/-)	MRI, FDG, Biosamples, fMRI, DTI, AV45
ADNI 2	350 (150/200)	MRI, FDG, Biosamples, fMRI, DTI, AV45

Different pre-processing methods for MRI images will be evaluated, by observing their effect on the final classification performance. To this end, we let a set of original MRI images undergo **four** different pre-processing pipelines. An outline of the general pre-processing steps is depicted in Figure 3.3, although the individual steps differs between the pipelines. The details of the pre-processing pipelines will be described in more detail under **Experiments and Results**.

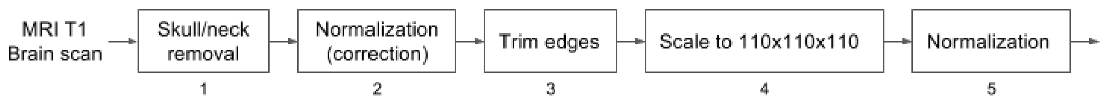


Figure 3.3: Overview of the MRI image preprocessing

3.2.2 Brain Tumor Classification

For Brain tumor classification, we use a dataset of MRI images acquired from the MICCAI BRaTS competition 2017 [42][43]. The dataset contains 210 images corresponding to High-grade glioma (HGG), and 75 to Low-grade glioma (LGG) (see Figure 1.4). Each image in the dataset is available weighted using T1, T1ce, T2 and FLAIR.
Only basic preprocessing steps were performed on the FLAIR MRI 3D images, namely **trimming** off obsolete black volumes in the edges, and **scaling** to uniform size (see Table 4.23).

4 Experiments and Results

This chapter is divided into different case studies (see Table 4.1), corresponding to different CNN architectures and pre-processing pipelines. We aim to investigate how the performance depends on the *dataset*, and different *hyperparameters*. In the last case study, we present our preliminary work on **brain tumor classification** into HGG and LGG.

Table 4.1: Summary of case studies

Case study	Mix. subj.	Architecture	Dataset	p_d	Best acc. of 3 runs	Avg. acc. of 3 runs
1 (AD)	yes	A	D_0	0.6	95.35%	95.03%
	yes	A	D_1	0.7	**98.74**%	**98.37**%
	yes	A	D_2	0.2	64.13%	-
	yes	A	D_3	0.7	61.47%	-
	no	A	D_1	0.6	**85.71**%	84.46%
	no	A	D_0	0.8	84.16%	-
2 (AD)	yes	B	D_0	0.5	92.50%	-
3 (AD)	yes	C	D_0	0.6	88.70%	-
4 (BT)	no	A	D_4	0.8	85.96%	-

4.1 Data Pre-processing

In the following two sections, we describe what type of pre-processing is done on the data used for Alzheimer's disease detection and for Brain tumor classification.

4.1.1 Alzheimer's Disease

To investigate the importance of the **data pre-processing**, we compare the performance when training and evaluating on **four** different datasets D_0, D_1, D_2, and D_3, which are obtained by applying different pre-processing pipelines to the same original MRI images (see details below). All data has been downloaded from ADNI (see Chapter 3), and has in some cases gone through some steps of pre-processing before we acquired the data.

A distinction is also made between when the datasets are partitioned randomly, i.e. mixed subjects, and when they are divided on patient level, so that images from the same patient are never in more than one of the training, validation, and test partitions.

In all experiments, the data is partitioned into *training* (60%), *validation* (20%), and *test* (20%) sets. The partitioning is random, with the exception of when considering the case of *separated subjects*. All training processes are limited to 150 epochs, and the model that is used for evaluation on the **test** data is the one that had the best **validation** performance during the training.

Table 4.2: Details of datasets used for AD detection

Dataset name		No. Subjects (M/F)	No. Images (M/F)	Images (%)
D_0, D_1	AD	199 (103/96)	600 (312/288)	50.08%
	NC	141 (75/66)	598 (331/267)	49.91%
	Total	340 (178/162)	1198 (643/555)	100%
D_2	AD	152 (77/75)	434 (204/230)	28.95%
	NC	331 (170/161)	1179 (617/562)	78.65%
	Total	483 (247/236)	1613 (821/792)	100%
D_3	AD	81 (-/-)	293 (-/-)	36.03%
	NC	92 (-/-)	520 (-/-)	63.97%
	Total	173 (-/-)	713 (-/-)	100%

4.1.1.1 Dataset D_0

As seen in Table 4.2, D_0 consists of 1198 T1 weighted MRI 3D images from a total of 340 subjects. The images in D_0 have undergone the following pre-processing steps (see Figure 4.1):

1. *recon-all* from the FreeSurfer software package[44] - includes Motion Correction and Conform, Non-Uniform intensity normalization, Talairach transform computation, skull and neck removal and others.
2. **Trim** i.e. find the smallest box that contains all useful pixels of an image, and remove obsolete black volumes from the edges.
3. **Scale** i.e. resample all images uniformly to the same resolution so that they contain the same number of pixels.
4. **Non-Uniform intensity normalization** using *nu_correct* from FreeSurfer
5. **Spatial normalization** using *flirt* from the FMRIB Software Library

Step 1 was performed by ADNI, i.e. already done when downloaded by us. Steps 2, 3, 4 and 5 were done during this project (see Figure 4.1), with the reason that we had observed other datasets (that for other reasons did not suit our needs) in the ADNI database that had exactly this pre-processing. These steps also required file format conversion between NIFTI and MINC using the MINC package, as well as re-orientation of the axes using *fslswapdim* from the FMRIB Software Library.

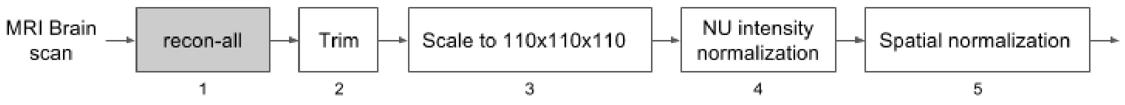


Figure 4.1: pre-processing pipeline for dataset D_0. Steps performed by ADNI are marked with gray background.

4.1.1.2 Dataset D_1

D_1 has the same original MRI images as D_0, and have similar pre-processing, except that steps 4 and 5 are left out (see Figure 4.2):

1. *recon-all* from the FreeSurfer software package[44]
2. **Trim**
3. **Scale**

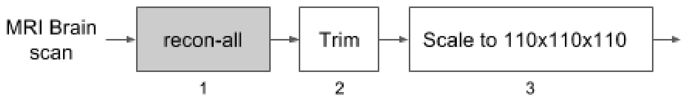


Figure 4.2: pre-processing pipeline for dataset D_1. Steps performed by ADNI are marked with gray background.

4.1.1.3 Dataset D_2

The following pre-processing steps have been applied to the images in D_2: (see Figure 4.3)

1. *GradWarp*
2. *B1 Correction*
3. *N3 Correction*
4. **Skull and Neck removal** using *mri_watershed* from the FreeSurfer software package.
5. **Trim**
6. **Scale**

Steps 1, 2, and 3 are done by ADNI, to whom we refer to regarding technical details of the pre-processing method.

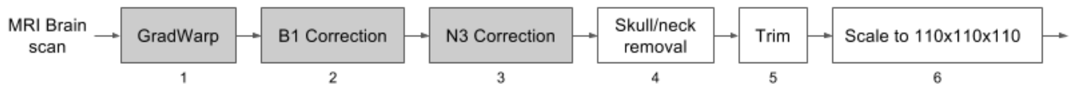


Figure 4.3: pre-processing pipeline for dataset D_2. Steps performed by ADNI are marked with gray background.

4.1.1.4 Dataset D_3

The images in D_3 have undergone the following pre-processing steps: (see Figure 4.4)

1. **Skull and Neck removal** using *mri_watershed* from the FreeSurfer software package.
2. **Trim**
3. **Scale**

Note that the dataset D_3 were original MRI, meaning that they were not preprocessed by ADNI.

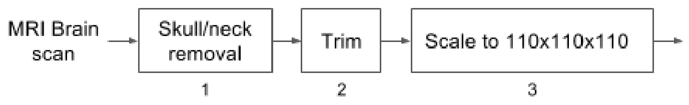


Figure 4.4: pre-processing pipeline for dataset D_3. Steps performed by ADNI are marked with gray background.

4.1.2 Brain Tumors

In our preliminary work on brain tumor classification, only simple pre-processing steps are done on the MRI images. This is described in the following section.

4.1.2.1 Dataset D_4

D_4 is the dataset used for brain tumor classification, and contains 286 FLAIR MRI 3D images, one for each patient. 210 of the images contain a high-grade tumor (HGG), and the remaining 75 low-grade tumors (LGG). Simple pre-processing steps were performed on this data: (see Figure 4.5)

1. **Trim**
2. **Scale**

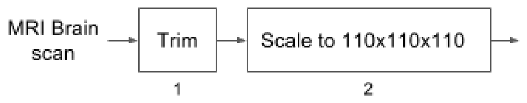


Figure 4.5: pre-processing pipeline for dataset D_4 of brain tumor MR images

4.2 Hardware and Software for Deep Learning

Deep learning requires a huge number of parameters to be optimized while training thus one of the most challenging consideration is the choice of hardware and software to reduce the training time as much as possible. Therefore the following is the description and specification of our choices for this project.

To design, implement and execute the architectures of this project, as well as training and testing, Cuda, TensorFlow, Python and Keras were chosen. To handle the demand for computational power that arises when training models of high complexity, Graphics Processing Unit (GPU) of model NVIDIA GTX Titan XP, alongside an Intel i7 Central Processing Unit (CPU) 64 bits, 3.40GHz and 128 GB of DDR4 RAM were chosen. More details about the hardware and software can be found in Table 4.3.

Table 4.3: Hardware and software specification

CPU	Intel-i7, 6800K, 3.40GHz, 64bits
GPU	NVIDIA Titan XP, 12GB Memory
OS	Ubuntu v.16.04.03, LTS, Xenial
RAM	DDR4 128GB
SSD	Samsung 64bits, 33MHz
Bus Controller	Intel-C610/X99 64bits, 33MHz
GPU Driver	NVIDIA v.375-, GLX v.4.5.0
Language	Python v.2.7.12, Tensorflow v.1.1.0, Cuda v.8.0.61, Keras v.2.0.4

4.3 Partitioning of Training/Validation/Test Sets

Some datasets consist of several MRI scans of each patient, which needs to be considered when evaluating the performance of the architectures. For some test cases, we make sure that the images from each patient are not part of both the data partitions used for training and test, i.e. that images from different partitions are never from the same subject. We use the term **separated subjects** when this is the case, and **mixed subjects** otherwise.

4.4 Case Study 1

In this section, we investigate the performance of CNN architecture A used for AD detection (see Table 3.1), using datasets with different pre-processing methods.
The hyperparameters (i.e. CNN structure, optimizer, p_d, η) for this architecture were found partly by inspiration from literature and previous works[11][16][19][20][21][24], but also through trial and error.

4.4.1 Dataset D_0

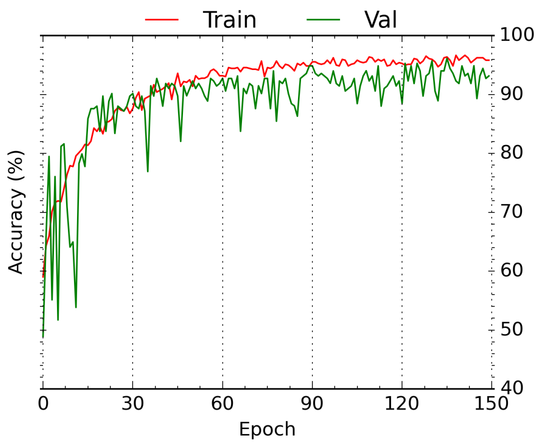
In this section, we investigate the performance of architecture A (see Table 3.1) on dataset D_0 shown in (see Table 4.2). The pre-processing steps of dataset D_0 are shown in Table 4.4. The performance when maintaining *separated subjects* will be compared to the one when using *mixed subjects*, and in both cases we will optimize the performance with respect to *dropout probability* and *learning rate*.

Table 4.4: pre-processing steps for dataset D_0

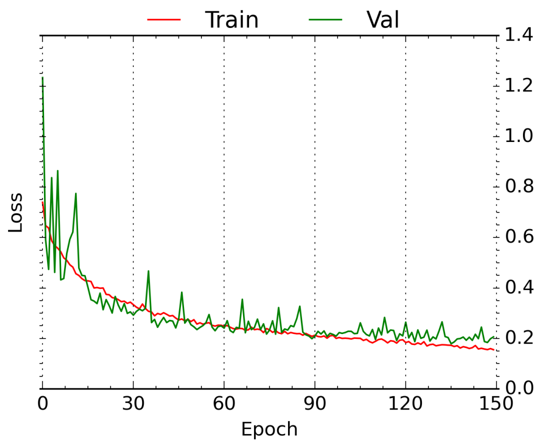
Step	Command	Effect
1.	recon-all	NU, Skull Striping, Intensity Normalization 2, Spherical Mapping, Smoothing, and others[44]
2.	Trim	Remove the black area
3.	mri_convert	Resize to 110*110*110
4.	fslswapdim	Reorient
5.	flirt	Spatial Normalization

- **Mixed Subjects**

The training process is visualized in Figure 4.6, where the performance (accuracy, as defined in 2.4.1.2, and loss[45]) is plotted. The resulting test accuracy is 95.35%, and the class sensitivities are shown in the *confusion matrix* (see Table 4.6).



(a) Accuracy



(b) Loss

Figure 4.6: Performance plotted during training of the best model achieved with structure A, trained and evaluated using dataset D_0 with mixed subjects. Learning rate $\eta = 0.01$, dropout probability $p_d = 0.6$. Test accuracy: 95.35%

Table 4.5: Confusion matrix of the performance on the test data, using the CNN parameters obtained in epoch 136 of Figure 4.6

		Prediction	
		NC	AD
Actual	NC	**0.941**	0.059
	AD	0.034	**0.966**

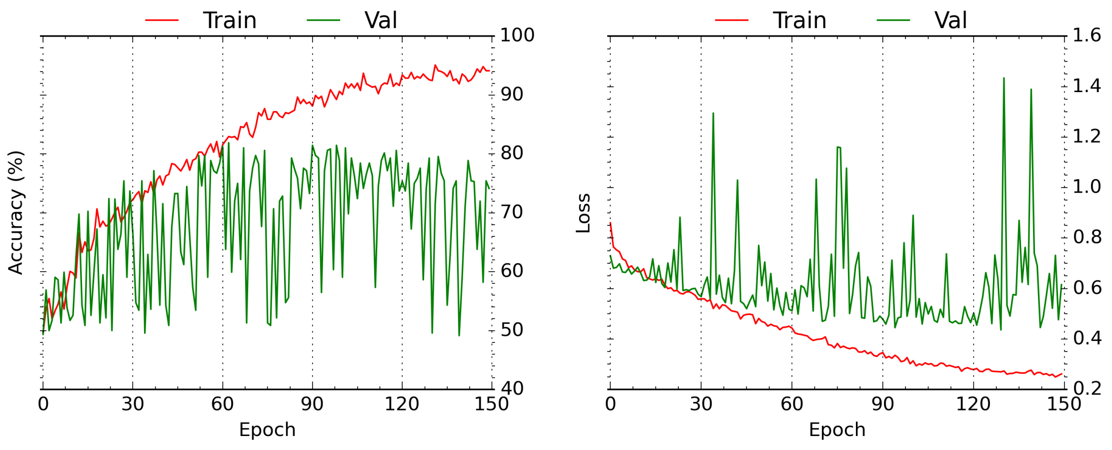
To verify that the training parameters are reliable, and hence the result reproducible, the training process is repeated three times with the same hyperparameters. The result is shown in Table 4.12

Table 4.6: Average test performance accuracy over 3 independent training runs for dataset A_0^{mix}.

Run	1	2	3	Average
Accuracy (%)	94.35	95.35	95.39	95.03

• Separated Subjects

Here, we maintain **separated subjects**, and thus prevent correlation between images in the train, validation, and test data. This method of evaluation should hence give a better measure on the potential classification performance of the model on input from a new, unknown subject.



(a) Accuracy **(b)** Loss

Figure 4.7: Performance plotted during training of the best model achieved with structure A, trained and evaluated using dataset D_0 with separated subjects. Learning rate $\eta = 0.01$, dropout probability $p_d = 0.8$. Test accuracy: 84.16%

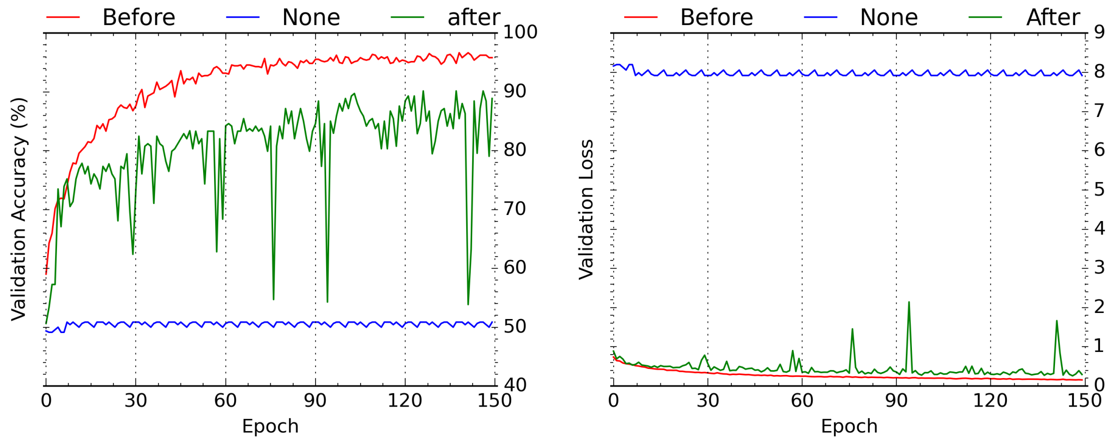
The performance of architecture A on dataset D_0 with separated subjects is, as seen in Figure 4.7, worse than on D_0 with mixed subjects. The test accuracy is 11.19 units of percentage lower in comparison, and visual inspection of Figure 4.7 reveals signs of overfitting early during the training. Due to limitation of computer power, we have not performed three runs for obtaining the average performance.

Table 4.7: Confusion matrix of the performance on the test data, using the CNN parameters obtained in epoch 63 of Figure 4.7

		Prediction	
		NC	AD
Actual	NC	**0.838**	0.162
	AD	0.156	**0.844**

4.4.1.1 Effect of Batch Normalization

There is no universally accepted convention on if the batch normalization layers should be applied *before* or *after* the activation function. In order to find out the optimum method for our circumstances, we investigate both cases, and also the effect of leaving out batch normalization completely. From Figure 4.8 we see that applying batch normalization *before* the activation function gives by far the most stable learning, and best performance. Applying batch normalization *after* the activation function shows slower and more unstable convergence, in addition to lower performance. Leaving out batch normalization seems to completely prevent the model from learning.



(a) Accuracy **(b)** Loss

Figure 4.8: Performance during training, using dataset A_0^{mix}. Learning rate $\eta = 0.01$ and $d_p = 0.6$.

4.4.1.2 Selection of Best Hyperparmeters

First, we to optimize the model performance over **learning rate** (η) and the **dropout probability** (p_d). This is done using **grid search**, which is an exhaustive search -style optimization method that evaluates all possible parameter combinations from sets of parameter values. The optimization is performed over the domains below (Eqs. 4.1, 4.2)

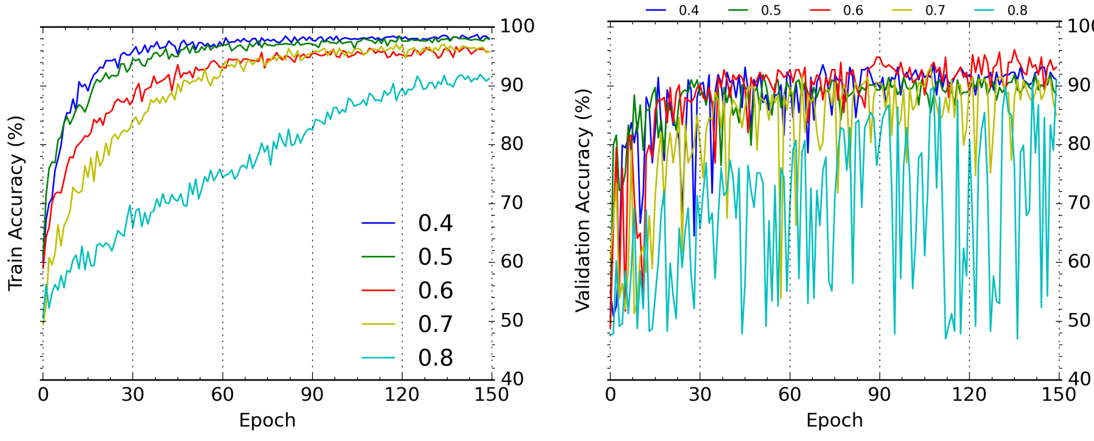
$$\eta \in \{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.02\} \tag{4.1}$$

$$d_p \in \{0.4, 0.5, \ldots, 0.8\} \tag{4.2}$$

The result of the grid search is shown in Table 4.8 and Figures 4.9, 4.10. We see in Table 4.8 that the best performance is achieved using learning rate $\eta = 0.01$ and dropout probability $p_d = 0.6$, resulting in an accuracy of 95.39%.

Table 4.8: Grid search result on dataset A_0^{mix}, optimizing w.r.t. learning rate η and dropout probability p_d

d_p \ η	0.0001	0.0005	0.001	0.005	0.01	0.02
0.4	71.96	85.35	85.51	92.88	91.63	92.88
0.5	69.87	81.58	87.86	91.63	91.21	90.37
0.6	66.94	73.22	81.58	93.30	**95.39**	90.79
0.7	67.36	71.96	76.15	93.30	88.70	91.21
0.8	48.11	72.38	69.3	82.42	85.77	91.63



(a) Train Accuracy **(b)** Validation Accuracy

Figure 4.9: Effect of different values of p_d for a fixed learning rate $\eta = 0.01$, on dataset D_0.

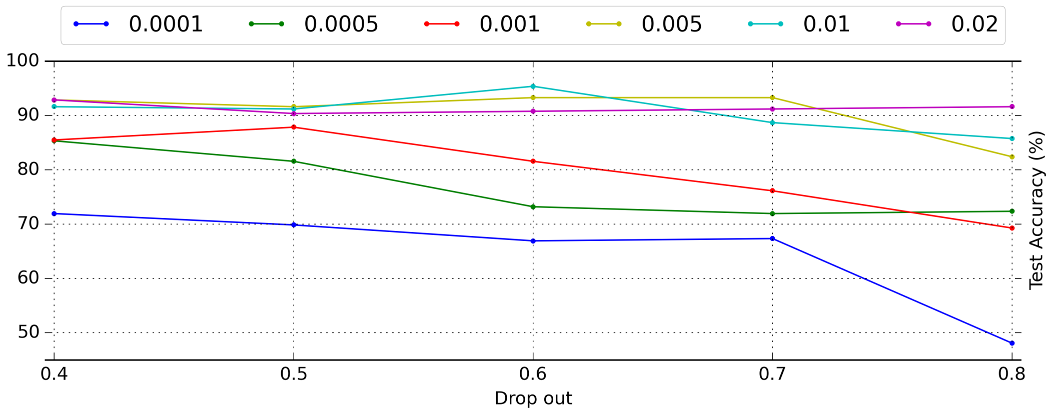


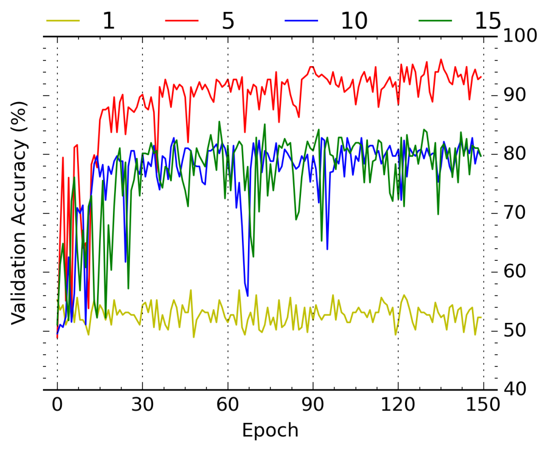
Figure 4.10: Test performance dependency on the dropout probability for fixed learning rates on dataset A_0^{mix}.

The time requirements for the grid search are shown in Table 4.9. One CNN training run takes 15 hours, and the entire grid search completed in approximately 18 days and 18 hours (450h).

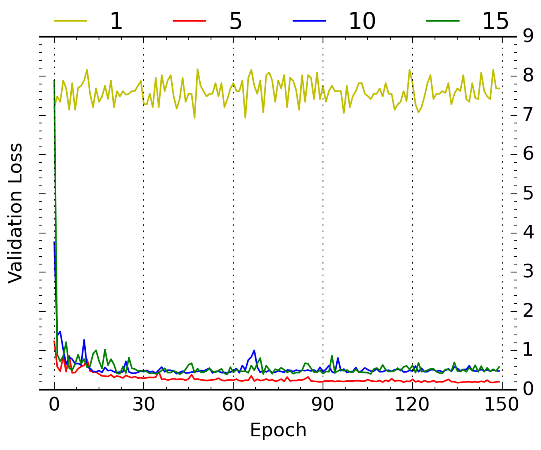
Table 4.9: Training time table for dataset D_0. Values mentioned in this table are approximated.

Dataset	No. Images	Image Resolution	Train (valid) time p. epoch	Total Time (150 epochs)	Grid Search
D_0	1198	110x110x110	312s (43s)	15h	450h

Next, we optimize the learning with respect to the **Batch size**. This is done by testing and comparing the performance of models trained with 4 different values, namely 1, 5, 10, and 15. From Figure 4.11 we see that from the values tested, a batch size of 5 results in the best validation performance.



(a) Accuracy



(b) Loss

Figure 4.11: Learning curves for different values batch sizes, dataset A_0^{mix}, $\eta = 0.01$, $p_d = 0.6$.

4.4.1.3 Discussion

Effect of separated subjects - We see a drop in performance when comparing the case of **separated subjects** with **mixed subjects**. This is not surprising, since when the subjects are mixed, images from different partitions (training, validation, and test) are likely to be correlated. This allows the network to learn features based on images from the training data, and then test based on other images from the same patients. Despite of this, the performance of the model on D_0 with separated subjects is promising, and encourages further testing and optimization.

Effect of batch normalization - From Figure 4.8b, we see that applying batch normalization **before** the activation function gives, by far, the best performance. This is to be expected, considering the activation used in this case, namely ReLU (see Theory). For input smaller than zero, the activation is the zero function, the output of which is no use in normalizing. For this domain it hence makes more sense to apply normalization before the activation, since otherwise it has no effect. Moreover, for input larger than zero, since ReLU then is the identity function, activation and normalization are commutative (i.e. the order of application irrelevant).

Optimal hyperparameters - From Table 4.8 it is obvious that the performance as a function of either the learning rate η or the dropout probability p_d (keeping one fixed) has many local optima, which is why hyperparameter optimization is tricky, and often require many trials, which for deep learning usually requires access to immense computer power. The resulting performance is of course also dependent on stochastic elements in the process, such as the *parameter initialization*, and the optimization using *Stochastic Gradient Descent*. It is therefore good practice to run the training several times, keeping the hyperparameters fixed, and taking all the results into account for evaluation.

4.4.2 Dataset D_1

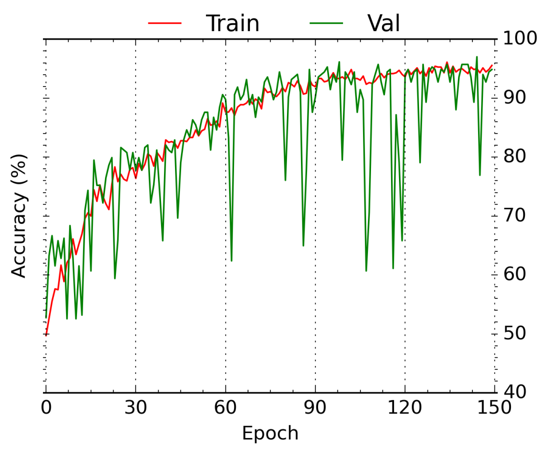
In this section, we evaluate the performance of architecture A trained and evaluated using dataset D_1, the pre-processing steps of which are shown in Table 4.10. Datasets D_1 and D_0 are the same, except for a few more additional pre-processing steps that have been done on D_1. The effect of these additional pre-processing steps will be investigated here, by comparing the performance of architecture A on both dataset D_1 and D_0. We will for this case also consider both *mixed subjects* and *separated subjects*, as well as completing one grid search to find the optimal training hyperparameters (see Table 4.13).

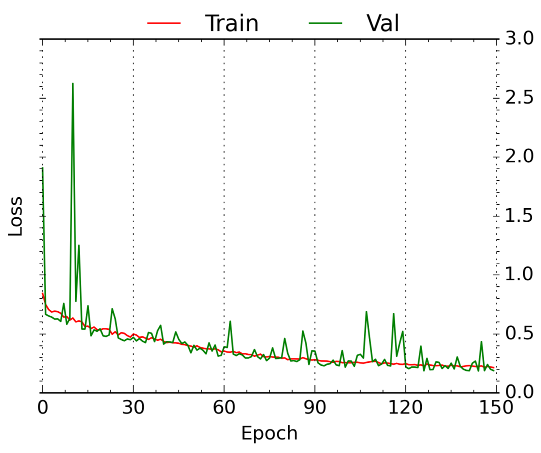
Table 4.10: pre-processing steps for dataset D_1

Step	Command	Effect
1.	recon-all	NU, Skull Striping, Intensity Normalization 2, Spherical Mapping, Smoothing, and others[44]
2.	Trim	Remove the black area
3.	mri_convert	Resize to 110*110*110

• Mixed Subjects

Grid search was used to find the optimal values of η and d_p (see Table 4.13). The training of the resulting optimal model is shown in Figure 4.12, where we see that using the same model A as above, but changing dataset to D_1 gives a higher resulting performance accuracy of 98.74%. The confusion matrix for the test result is shown in Table 4.11.





(a) Accuracy **(b)** Loss

Figure 4.12: Performance plotted during training of the best model achieved with structure A, trained and evaluated using dataset A_1^{mix}. Learning rate $\eta = 0.01$, dropout probability $p_d = 0.7$. Test accuracy: 98.74%

In the confusion matrix (Table 4.11), we see the class specific accuracies, i.e. the test performance on each class. Here we see that a reasonably high detection rate of NC, namely approx. 98.37%, but an astoundingly good performance of 100% detection rate of AD. Practically, this means that every image from a patient with Alzheimer's disease was flagged by our system, and that approx. 2.4% of healthy people

Table 4.11: Confusion matrix of the performance on the test data, using the CNN parameters obtained in epoch 145 of Figure 4.12

		Prediction	
		NC	AD
Actual	NC	**0.9756**	0.0244
	AD	0.00	**1.00**

As for dataset D_0, we establish the reliability of the used hyperparameters by performing three CNN training runs. The result is shown in Table 4.12, where we see that the resulting performance is repeatable, with an expected reasonable deviation in the test accuracy.

Table 4.12: Average test performance accuracy over 3 independent training runs for dataset A_1^{mix}.

Run	1	2	3	Average
Accuracy (%)	**98.74**	98.48	97.90	98.37

As mentioned above, we perform a grid search also for dataset D_1, where we seek the optimal values for the hyperparameters η (learning rate) and p_d (dropout probability). The result is shown in Table 4.13, where we see that the best performance was achieved with $\eta = 0.01$, which is the same as the optimum for dataset D_0, and $p_d = 0.7$ (compared to the optimum $p_d = 0.6$ for dataset D_0).

Table 4.13: Grid search result on dataset A_1^{mix}, optimizing w.r.t. learning rate η and dropout probability p_d

d_p \ η	0.0001	0.0005	0.001	0.005	0.01	$\eta = 0.02$
0.4	73.64	88.28	92.46	95.81	95.39	95.39
0.5	70.71	88.28	90.79	95.81	94.56	97.7
0.6	61.92	77.40	88.28	96.23	97.70	92.50
0.7	71.96	71.12	76.98	93.72	**97.90**	97.70
0.8	53.55	68.20	71.54	93.72	92.88	94.97

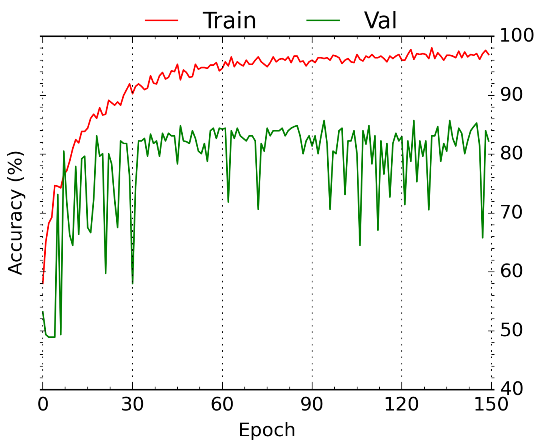
The time requirements for the grid search is the same as for dataset D_0 (see Table 4.8), and are shown here in Table 4.14. One CNN training run takes 15 hours, and the entire grid search completed in approximately 18 days and 18 hours.

Table 4.14: Training time table for dataset D_0. Values mentioned in this table are approximated.

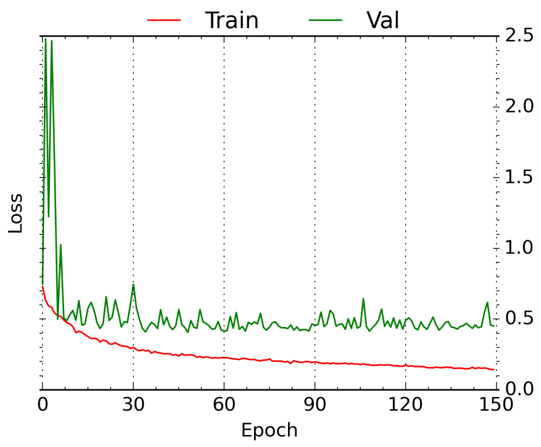
Dataset	No. Images	Image Resolution	Train (valid) time p. epoch	Total Time (150 epochs)	Grid Search
D_0	1198	110x110x110	312s (43s)	15h	450h

• Separated Subjects

When evaluating model A on dataset D_1 with separated subjects, we perform no optimization with respect to η, and use a fixed value $\eta = 0.01$, since this has resulted in the best performance for all of the above case studies. Instead, the effect of the learning rate η for this model and dataset is investigated. The training that resulted in the best model is shown in Figure 4.13, which performed with a 85.71% accuracy on the test data. In this case study, instead of using grid search, different sets of values of η is tried, for two different step lengths (see Table 4.16 and 4.18). In the first case, i.e. using step length 0.05, the best performance achieved was 84.87% for $p_d = 0.85$ (Table 4.16). However, using a larger step length 0.1, and hence covering a larger range with lower precision, resulted in an optimal performance of 85.71% for $d_p = 0.6$.



(a) Accuracy



(b) Loss

Figure 4.13: Performance plotted during training of the best model achieved with structure A, trained and evaluated using dataset A_1^{sep}. Learning rate $\eta = 0.01$, dropout probability $p_d = 0.6$. Test accuracy: 85.71%

The class specific test performance (class sensitivity) is shown as a confusion matrix in Table 4.15.

Table 4.15: Confusion matrix of the performance on the test data, using the CNN parameters obtained in epoch 95 of Figure 4.13

		Prediction	
		NC	AD
Actual	NC	**0.864**	0.136
	AD	0.167	**0.833**

The first optimization with respect to p_d is shown in Table 4.16, where we see that the best performance (84.87%) is achieved using $p_d = 0.85$. In Table 4.17, we show the result from training using $p_d = 0.85$ three times.

Table 4.16: Optimization of the test accuracy w.r.t. p_d with step size 0.05, with a fixed learning rate $\eta = 0.01$ on dataset D_1 with separated subjects.

p_d	0.60	0.65	0.70	0.75	0.80	0.85	0.9
Test Accuracy (%)	82.35	83.61	81.51	79.83	81.51	**84.87**	76.50

Table 4.17: Average test performance accuracy over 3 independent training runs for dataset A_1^{Sep}.

Run	1	2	3	Average
Accuracy (%)	84.87	84.35	85.29	84.46

The second optimization, where we use a larger step size of 0.1, gave a result inconsistent with the first optimization, namely that the optimal performance is achieved using $p_d = 0.6$, which in this run gave an accuracy of 85.71%, compared to 82.35% which was the corresponding result from the first optimization.

Table 4.18: Optimization of the test accuracy w.r.t. p_d with step size 0.1, with a fixed learning rate $\eta = 0.01$ on dataset D_1 with separated subjects.

p_d	0.4	0.5	0.6	0.7	0.8
Test Accuracy (%)	80.67	81.93	**85.71**	84.30	81.93

4.4.2.1 Discussion

Impact of pre-processing - Using dataset D_1 for training and testing model A gives a higher performance of 97.9% compared to using dataset D_0. Several runs with fixed hyperparameter (see Table 4.17) confirms that this is not only by chance, hence the only possible causing variable is the dataset. Datasets D_0 and D_1 differs with a few steps in the pre-processing (recall Methods), the effect of which must have affected the data in such a way so that the features used for detecting AD are less distinguishable.

separated subjects - We see for dataset D_1, similarly as for D_0, that the performance drops when partitioning with *separated subjects*. In the initial search for the best dropout probability p_d, we found that the optimal value is $p_d = 0.85$, with a corresponding test accuracy of 84.87%. However, during the search with a larger time step, we found that $p_d = 0.6$ allows training of a model that performs with a test accuracy of 85.71%, which is odd, considering that during the first search, this dropout probability resulted in a model with only 82.35% test accuracy. This is most likely caused by the stochastic elements of the learning process, which again demonstrates the importance of performing several training runs for fixed hyperparmeters before drawing a conclusion.

4.4.3 Impact of Data pre-processing on CNN Performance

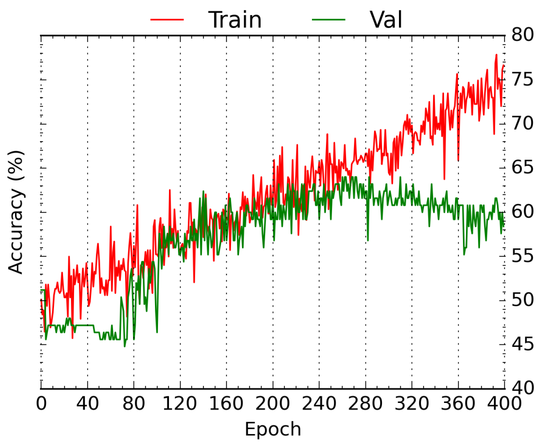
In this section, we further investigate the impact of data pre-processing by training and evaluating our best architecture A on datasets D_2 and D_3.

4.4.3.1 Dataset D_2

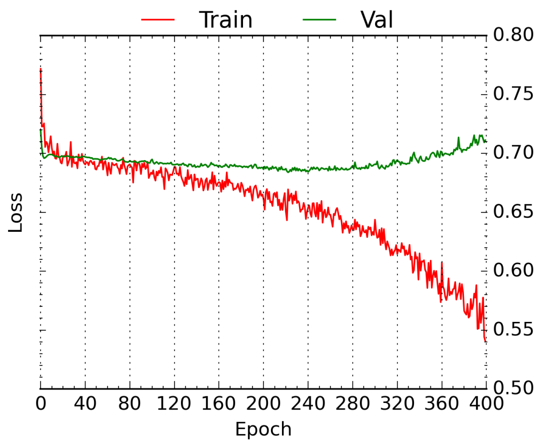
The usefulness of architecture A on dataset D_2 (see Table 4.2) is investigated. The The hyperparameter optimization for this dataset was done mainly through trial and error of many, as we at this stage in the project were not certain of within which intervals hyperparameters are more likely to result in the best performing model. This resulted in $\eta = 0.0001$ and $p_d = 0.2$, for which the corresponding learning process is visualized in Figure 4.14a. In mentioned Figure we note that (1) the validation accuracy never exceeds 65% and (2) the validation loss is decreasing until epoch 270, after which it has an increasing trend for the rest of the training duration. Accordingly with *early stopping*, the resulting model after the 270th epoch is hence evaluated on the test data, resulting in a classification performance of 64.13%.

Table 4.19: Pre-processing steps for dataset D_2

Step	Command	Effect
1.	proc_ADNI_script	GradWarp
2.	proc_ADNI_script	B1 Correction
3.	nu_correct	N3
4.	proscale_nii	Scaled
5.	Trim	Remove the black area
6.	mri_convert	Resize to 110*110*110



(a) Accuracy



(b) Loss

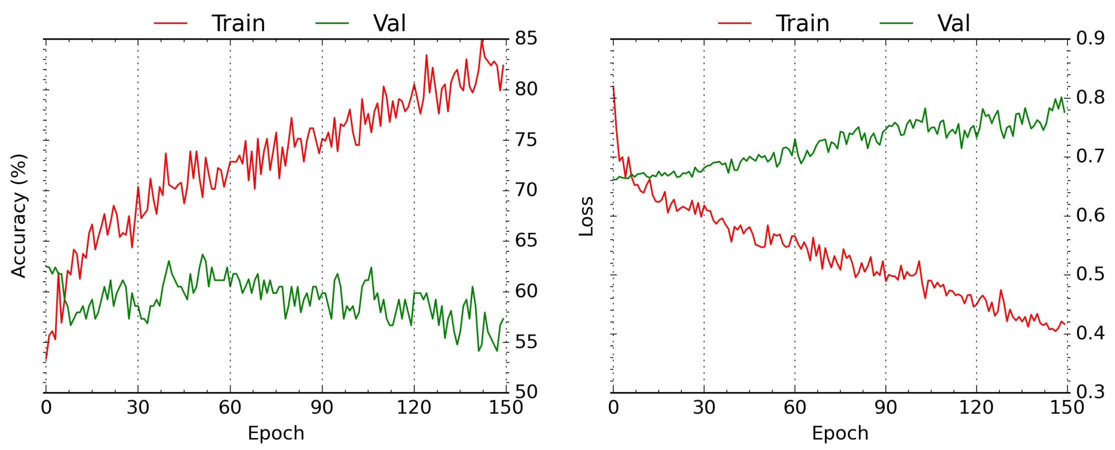
Figure 4.14: Learning curve for model A on dataset B^{mix}, with $\eta = 0.0001$ and $p_d = 0.2$. Test accuracy: 64.13%

4.4.3.2 Dataset D_3

We investigate the performance of model A on dataset D_3. The hyperparameter optimization for this case is as for dataset D_2, namely by trial and error, which resulted in $\eta = 0.01$ and $p_d = 0.7$. From Figure 4.15 we see similarities to the performance on dataset D_2, namely that the accuracy at which the model performs is limited. However in this case, by inspecting the *loss* curve, we see barely any improvement. The test accuracy of the resulting model is 61.47%.

Table 4.20: Pre-processing steps for dataset D_3

Step	Command	Effect
1.	mri_watershed	Skull removed
5.	Trim	Remove the black area
6.	mri_convert	Resize to 110*110*110



(a) Accuracy **(b)** Loss

Figure 4.15: Best result of training model A on dataset D_3 with mixed subjects, $\eta = 0.01$ and $p_d = 0.7$. Test accuracy: 61.47%.

4.4.3.3 Discussion

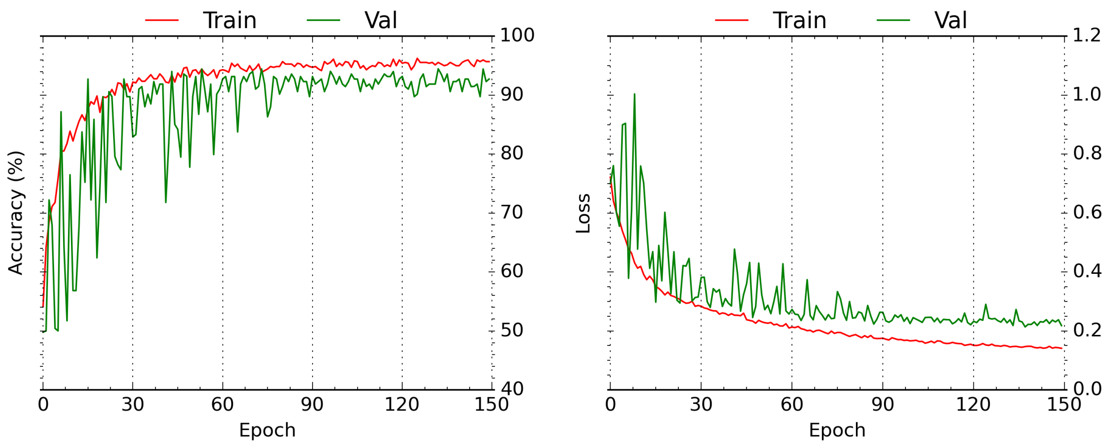
Impact of pre-processing - The performance on dataset D_2 and D_3 compared to D_0 or D_1 demonstrates the importance of what kind of data pre-processing to be used. For dataset D_2, in Figure 4.14, one can observe signs of successful learning until epoch 270, where the validation loss starts to increase. The plot shows clear signs of overfitting, which was not successfully prevented during this project by using the usual methods (I.e. data augmentation, dropout, batch normalization, weight decay, simpler structure). It is possible that a more complex model is required to successfully extract the desired features, but since overfitting is already a problem here, this would most likely require more (or augmented) training data.

4.5 Case Study 2

In order to have a comparison to model A, we investigate the performance of the similar architecture B for AD detection, defined in Table 3.2.

4.5.1 Dataset D_0

The performance of architecture B on dataset D_0 is investigated. The result of optimizing with respect to the dropout probability p_d is shown in Table 4.21, which shows the best performance for $p_d = 0.5$. The corresponding learning curve is shown in (Figure 4.16). In the figure, we see stable validation performance improvement during training. Note, however, that the the performance is not as good as model A on the same dataset.



(a) Accuracy **(b)** Loss

Figure 4.16: Training of the best model on dataset A_0^{mix}, $\eta = 0.01$, $p_d = 0.5$. Test accuracy: 92.50%

Table 4.21: Optimization of the performance of architecture B w.r.t. p_d, for a fixed learning rate $\eta = 0.01$, using dataset A_0^{mix}.

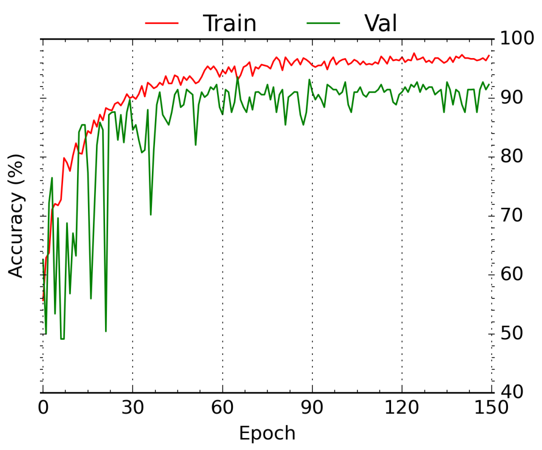
p_d	0.4	0.5	0.6	0.7	0.8
Test Accuracy (%)	92.45	**92.50**	89.95	91.63	87.2

4.6 Case Study 3

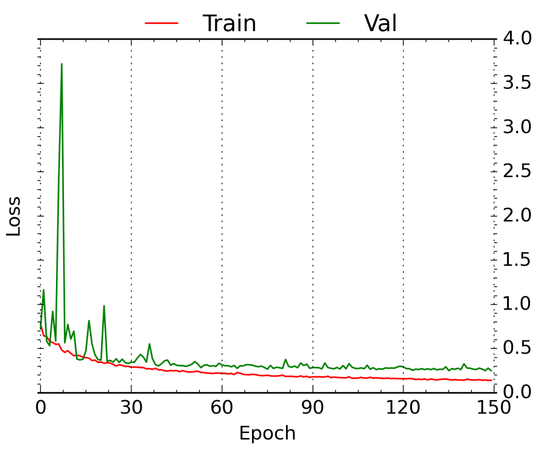
In addition to architecture B, we introduce architecture C (see Table 3.3) to provide a comparison to architecture A.

4.6.1 Dataset D_0

We investigate the performance of architecture C on dataset D_0 with mixed subjects. No type of optimization was done for this case study, instead the same hyperparameters as in the case of model A were used, i.e. $p_d = 0.6$. The result is shown in Figure 4.17, and the final test accuracy was 88.70%.



(a) Accuracy



(b) Loss

Figure 4.17: The best result of architecture C trained on dataset A_0^{mix}, for $\eta = 0.01$ and $p_d = 0.6$. Test accuracy: 88.70%

4.7 Case Study 4 for Brain Tumor Classification

In this case study, we aim to achieve **brain tumor classification** into HGG and LGG using the 3D-CNN architecture A (see Table 3.1). The system is trained and evaluated on a dataset of 3D FLAIR MRI images distributed according to Table 4.22, preprocessed according to 4.23.

4.7.1 Dataset D_4

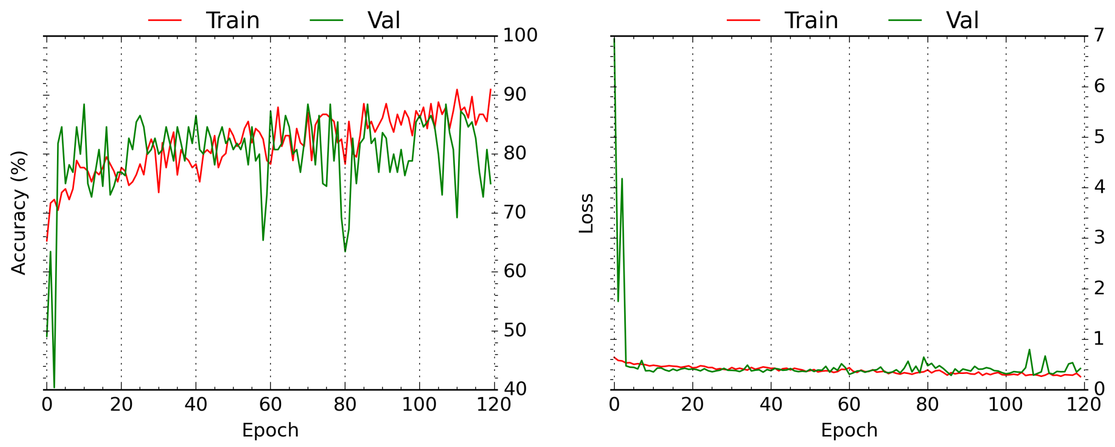
During the training of architecture A (see Table 3.1), the learning rate is $\eta = 0.01$ since this seemed to be a universally good choice based on the optimization results on AD detection performance (see e.g. Tables 4.8, 4.13). The dropout probability p_d was varied during initial testing, and the best performance seemed to be achieved using $p_d = 0.8$. The resulting learning curve is visualized in Figure 4.18, with a resulting test accuracy of 85.96%, and the class specific performance (class sensitivities) are shown as a **confusion matrix** in Table 4.24.

Table 4.22: Dataset D_4 specification

Type	No. Images	Images Distribution
HGG	210	73.68%
LGG	75	26.31%

Table 4.23: pre-processing steps for BRaTS dataset

Step	Command	Effect
1.	Trim	Remove the black area
2.	mri_convert	Resize to 110*110*110



(a) Loss **(b)** Accuracy

Figure 4.18: Performance during training, using Brain Tumor dataset. Learning rate $\eta = 0.05$ and $d_p = 0.8$. The test accuracy was 85.96%.

Table 4.24: Confusion matrix of the performance on the test data, using the CNN parameters obtained in epoch 87 of Figure 4.18

		Prediction	
		HGG	LGG
Actual	HGG	0.88	0.12
	LGG	0.24	0.76

4.8 Summary

We have evaluated three different CNN architectures, whereof A is the one that achieved the best overall performance (see Table 4.1). Despite the low number of neurons in the fully connected layer, and the fact that the model was not pretrained, an accuracy of 98.74% and AD detection rate 100% was reached by training model A using dataset D_1 with mixed subjects. The optimal *learning rate* and *dropout probability* were found to be $\eta = 0.01$ and $p_d = 0.6$.

A drop in performance to an accuracy of 85.71% was observed when using dataset D_1 with separated subjects. It has been shown that architecture A cannot successfully *learn* on either of datasets D_2 and D_3, the pre-processing of which differs significantly from the one of dataset D_0.

The significance of **batch size** is shown in Figure 4.11, the optimal value of which was empirically established to be five.

The effect of different orders of application of **batch normalization** and **activation function** is investigated. It is shown in Figure 4.8 that the best performance is achieved by applying batch normalization *before* the activation function, for which is also argued theoretically in the discussion appended in the corresponding section.

The results of using architecture B and C were evaluated only on dataset D_0, and showed slightly reduced performance compared to model A.

The performance of architecture A on dataset D_4 containing FLAIR MRI images of glioma patients was evaluated. The preliminary results achieved for high-grade/low-grade glioma classification shows a moderately high classification accuracy of approx. 86%.

4.9 Comparison with Existing Works

We conjecture that our system is *state-of-art*, mainly because of the system performance and secondly the large size of the dataset. Our system has been trained and evaluated on a dataset where the classes are balanced (50% AD and NC) with a higher number of patients and images than most other related works, and the data pre-processing is relatively simple and straight-forward in comparison.

Table 4.25: Related Work Results Comparison with our work. Abbreviation used are : Deep Learning(DL), Sparse AutoEncoders(SAE), Adaptable CNN(ACNN).

References	No. Subjects (NC/AD)	No. Images (NC/AD)	Method	Accuracy
Sarrafa,et al.[20]	302 (91/211)	-	DL-CNN	98.84
Payan, et al.[21]	432 (232/200)	1510 (755/755)	SAE-CNN	95.39
Hosseini, et al.[24]	140 (70/70)	-	3D-ACNN	97.6
Liu, et al.[25]	142 (77/65)	-	Auto-Encode	88.57
Our result	340 (140/200)	1198 (598/600)	3D-CNN	98.70

5
Conclusion

Our results show that it is possible to create a 3D-CNN based system that with high accuracy classifies T1 weighted MRI brain scans to be either Alzheimer's Disease or Normal Control, with requirements on a relatively simple and straight-forward method of preprocessing of the dataset.

We have optimized the parameters for the training process of our deep CNN architecture, which resulted in a model that achieved an accuracy of 98.74% on an independent test set. The model performed with an accuracy of 97.56% on the NC test images, which means that 2.44% of the NC images were incorrectly classified as AD (false alarm), but performed with an astounding 100% on the AD images. This means that every image from a patient with Alzheimer's disease was flagged by our system. This is a very desirable behaviour, since it is much more serious to incorrectly classify a sick patient as healthy, rather than the other way around.

This work shows the potential of applying artificial intelligence in medicine (neurology) and the benefits it would imply, including greatly reduced time and cost for establishing a diagnose, higher chance of a physician correctly diagnosing a patient when using AI as assistance, and thus overall earlier disease detection. This is vital when it comes to e.g. brain tumor classification, since the treatment, that must be begun the soonest possible after onset, is very dependent on the type of tumor. Our preliminary results show that using CNN classification systems have potential to perform this type of classification. A system with higher accuracy would not only lead to a shorter and less expensive diagnosing process, but more importantly also a non-invasive patient examination, namely only processing of an MRI brain scan.

5.1 Future Work

Further investigation of the effect of different structures and hyperparameters is needed, and it would not be surprising if it led to a classifier with a higher accuracy. The performance for separated subjects has room for improvement, but this is likely to require a larger dataset. Moreover, further testing and investigation regarding data preprocessing, network architecture and other hyperparameters for glioma classification is needed.

Bibliography

[1] WHO Dementia. fact sheet n 362. geneva, switzerland: World health organization, 2015.

[2] Jeffrey L Cummings, Travis Morstorf, and Kate Zhong. Alzheimer's disease drug-development pipeline: few candidates, frequent failures. *Alzheimer's research & therapy*, 6(4):37, 2014.

[3] Mei Sian Chong and Suresh Sahadevan. Preclinical alzheimer's disease: diagnosis and prediction of progression. *The Lancet Neurology*, 4(9):576–579, 2005.

[4] Kenechukwu Monplaisir Monplaisir. *Effect of oil palm phenolics on beta amyloid deposition in cholesterol induced rat model of Alzheimer's disease: Histological evidence*. PhD thesis, Wayne State University, 2016.

[5] Michael H Repacholi. Low-level exposure to radiofrequency electromagnetic fields: Health effects and research needs. *Bioelectromagnetics*, 19(1):1–19, 1998.

[6] David N Louis, Arie Perry, Guido Reifenberger, Andreas Von Deimling, Dominique Figarella-Branger, Webster K Cavenee, Hiroko Ohgaki, Otmar D Wiestler, Paul Kleihues, and David W Ellison. The 2016 world health organization classification of tumors of the central nervous system: a summary. *Acta neuropathologica*, 131(6):803–820, 2016.

[7] Nicolas R Smoll, Oliver P Gautschi, Bawarjan Schatlo, Karl Schaller, and Damien C Weber. Relative survival of patients with supratentorial low-grade gliomas. *Neuro-oncology*, 14(8):1062–1069, 2012.

[8] Fonnet E Bleeker, Remco J Molenaar, and Sieger Leenstra. Recent advances in the molecular understanding of glioblastoma. *Journal of neuro-oncology*, 108(1):11–27, 2012.

[9] DRGHR Williams and Geoffrey Hinton. Learning representations by back-propagating errors. *Nature*, 323(6088):533–538, 1986.

[10] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[11] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[13] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[14] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[15] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[18] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[19] Li Deng. Three classes of deep learning architectures and their applications: a tutorial survey. *APSIPA transactions on signal and information processing*, 2012.

[20] Saman Sarraf, John Anderson, Ghassem Tofighi, et al. Deepad: Alzheimer's disease classification via deep convolutional neural networks using mri and fmri. *bioRxiv*, page 070441, 2016.

[21] Adrien Payan and Giovanni Montana. Predicting alzheimer's disease: a neuroimaging study with 3d convolutional neural networks. *arXiv preprint arXiv:1502.02506*, 2015.

[22] ALZHEIMER'S DISEASE NEUROIMAGING INITIATIVE. About.

[23] Ashish Gupta, Murat Ayhan, and Anthony Maida. Natural image bases to represent neuroimaging data. In *International Conference on Machine Learning*, pages 987–994, 2013.

[24] Ehsan Hosseini-Asl, Robert Keynton, and Ayman El-Baz. Alzheimer's disease diagnostics by adaptation of 3d convolutional network. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 126–130. IEEE, 2016.

[25] Siqi Liu, Sidong Liu, Weidong Cai, Sonia Pujol, Ron Kikinis, and Dagan Feng. Early diagnosis of alzheimer's disease with deep learning. In *Biomedical Imaging (ISBI), 2014 IEEE 11th International Symposium on*, pages 1015–1018. IEEE, 2014.

[26] Neural Networks and Machine Learning. Info 2040/cs 2850/econ 2040/soc 2090.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4):185–196, 1993.

[29] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[30] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[32] David C Plaut et al. Experiments on learning by back propagation. 1986.

[33] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[34] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[35] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[36] James Bergstra, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, Ian Goodfellow, Arnaud Bergeron, Yoshua Bengio, and Pack Kaelbling. Theano: Deep learning on gpus with python. 2011.

[37] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, Idiap, 2002.

[38] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[39] François Chollet et al. Keras: Deep learning library for theano and tensorflow. *URL: https://keras. io/k*, 2015.

[40] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[41] ALZHEIMER'S DISEASE NEUROIMAGING INITIATIVE. adni-data-inventory.

[42] Bjoern H Menze, Andras Jakab, Stefan Bauer, Jayashree Kalpathy-Cramer, Keyvan Farahani, Justin Kirby, Yuliya Burren, Nicole Porz, Johannes Slotboom, Roland Wiest, et al. The multimodal brain tumor image segmentation benchmark (brats). *IEEE transactions on medical imaging*, 34(10):1993–2024, 2015.

[43] Mina Rezaei, Konstantin Harmuth, Willi Gierke, Thomas Kellermeier, Martin Fischer, Haojin Yang, and Christoph Meinel. Conditional adversarial network for semantic segmentation of brain tumor. *arXiv preprint arXiv:1708.05227*, 2017.

[44] FreeSurfer. Recon-all.

[45] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

A

Appendix 1

Table A.1: Detailed description of 3D-CNN architecture *A*. Abbreviations used in this table: Batch Normalization (BN), Fully Connected (FC).

ConvNet Structure				
Layer Type	Kernel,Filter-Stride/Neuron	Input Size	Output Size	Pad
Input	-	110,110,110	-	-
Conv1,BN,Relu	7,7,7,64-(2)	110,110,110	52,52,52,64	NO
Conv2,BN,Relu	3,3,3,64-(1)	52,52,52,64	50,50,50,64	NO
Conv3,BN,Relu,MaxPool	3,3,3,128-(1)	50,50,50,64	24,24,24,128	NO
Conv4,BN,Relu,MaxPool	3,3,3,128-(1)	24,24,24,128	11,11,11,128	NO
Conv5,BN,Relu,MaxPool	3,3,3,128-(1)	11,11,11,128	4,4,4,128	NO
Flatten	-	4,4,4,128	8192	-
FC1,BN,Relu	256	8192	256	-
FC2,BN,Relu	256	256	256	-
FC3,BN,Relu,Softmax	2	256	2	-

Table A.2: Detailed description of 3D-CNN architecture *B*. Abbreviations used in this table: Batch Normalization (BN), Fully Connected (FC).

ConvNet Structure				
Layer Type	Kernel,Filter-Stride/Neuron	Input Size	Output Size	Pad
Input	-	110,110,110	-	-
Conv1,BN,Relu	3,3,3,64-(2)	110,110,110	54,54,54,64	NO
Conv2,BN,Relu	3,3,3,64-(1)	54,54,54,64	52,52,52,64	NO
Conv3,BN,Relu,MaxPool	3,3,3,128-(1)	52,52,52,64	25,25,25,128	NO
Conv4,BN,Relu,MaxPool	3,3,3,128-(1)	25,25,25,128	11,11,11,128	NO
Conv5,BN,Relu,MaxPool	3,3,3,128-(1)	11,11,11,128	4,4,4,128	NO
Flatten	-	4,4,4,128	8192	-
FC1,BN,Relu	256	8192	256	-
FC2,BN,Relu	256	256	256	-
FC3,BN,Relu,Softmax	2	256	2	-

Table A.3: Detailed description 3D-CNN architecture *C*. Abbreviations used in this table: Batch Normalization (BN), Fully Connected (FC).

ConvNet Structure				
Layer Type	Kernel,Filter-Stride/Neuron	Input Size	Output Size	Pad
Input	-	110,110,110	-	-
Conv1,BN,Relu	7,7,7,64-(2)	110,110,110	52,52,52,64	NO
Conv2,BN,Relu	3,3,3,64-(1)	52,52,52,64	50,50,50,64	NO
Conv3,BN,Relu,MaxPool	3,3,3,128-(1)	50,50,50,64	24,24,24,128	NO
Conv4,BN,Relu,MaxPool	3,3,3,128-(1)	24,24,24,128	11,11,11,128	NO
Flatten	-	4,4,4,128	170368	-
FC1,BN,Relu	256	170368	256	-
FC2,BN,Relu	256	256	256	-
FC3,BN,Relu,Softmax	2	256	2	-