EFFICIENT DEEP LEARNING OF 3D STRUCTURAL BRAIN MRIS FOR MANIFOLD LEARNING AND LESION SEGMENTATION WITH APPLICATION TO MULTIPLE SCLEROSIS

by

TOM BROSCH

Dipl.-Ing., Otto-von-Guericke-Universität Magdeburg, 2010

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES (Biomedical Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

July 2016

© Tom Brosch, 2016
Abstract

Deep learning methods have shown great success in many research areas such as object recognition, speech recognition, and natural language understanding, due to their ability to automatically learn a hierarchical set of features that is tuned to a given domain and robust to large variability. This motivates the use of deep learning for neurological applications, because the large variability in brain morphology and varying contrasts produced by different MRI scanners makes the automatic analysis of brain images challenging.

However, 3D brain images pose unique challenges due to their complex content and high dimensionality relative to the typical number of images available, making optimization of deep networks and evaluation of extracted features difficult. In order to facilitate the training on large 3D volumes, we have developed a novel training method for deep networks that is optimized for speed and memory. Our method performs training of convolutional deep belief networks and convolutional neural networks in the frequency domain, which replaces the time-consuming calculation of convolutions with element-wise multiplications, while adding only a small number of Fourier transforms.

We demonstrate the potential of deep learning for neurological image analysis using two applications. One is the development of a fully automatic multiple sclerosis (MS) lesion segmentation method based on a new type of convolutional neural network that consists of two interconnected pathways for feature extraction and lesion prediction. This
Abstract

allows for the automatic learning of features at different scales that are optimized for accuracy for any given combination of image types and segmentation task. Our network also uses a novel objective function that works well for segmenting underrepresented classes, such as MS lesions. The other application is the development of a statistical model of brain images that can automatically discover patterns of variability in brain morphology and lesion distribution. We propose building such a model using a deep belief network, a layered network whose parameters can be learned from training images. Our results show that this model can automatically discover the classic patterns of MS pathology, as well as more subtle ones, and that the parameters computed have strong relationships to MS clinical scores.
Preface

This thesis is primarily based on two journal papers, three conference papers, and one book chapter, resulting from the collaboration between multiple researchers. In all publications, the contribution of the author was in developing, implementing, and evaluating the method. All co-authors contributed to the editing of the manuscripts.

Parts of the introduction to deep learning in Chapter 2 have been published in:


The contribution of the author was in writing the introduction of the book chapter. Y. Yoo and R. Tam wrote the remaining sections of the chapter. All co-authors contributed to the editing of the manuscript.

The study described in Chapter 3 has been published in:

The contribution of the author was in developing, implementing, and evaluating the method. R. Tam helped with his valuable suggestions in improving the methodology.

A study described in Chapter 4 has been published in:


The contribution of the author was in developing, implementing, and evaluating the method. L.Y.W. Tang helped with the evaluation of other lesion segmentation methods. A. Traboulsee and D.K.B. Li provided the data and clinical input. Y. Yoo, L.Y.W. Tang, and R. Tam helped with their valuable suggestions in improving the methodology.

A study described in Chapter 5 has been published in:


The contribution of the author was in developing, implementing, and evaluating the method. R. Tam helped with his valuable suggestions in improving the methodology.
A study described in Chapter 5 has been published in:


The contribution of the author was in developing, implementing, and evaluating the method. A. Traboulsee and D.K.B. Li provided the data and clinical input. Y. Yoo and R. Tam helped with their valuable suggestions in improving the methodology.
Table of Contents

Abstract ................................................................. ii
Preface ................................................................. iv
Table of Contents ...................................................... x
List of Tables .......................................................... xi
List of Figures .......................................................... xiii
List of Abbreviations .................................................. xv
Acknowledgements ..................................................... xx
Dedication ............................................................... xxii

1 Introduction ......................................................... 1
   1.1 Motivation ......................................................... 1
   1.1.1 A short introduction to multiple sclerosis .................. 1
   1.1.2 Measuring disease state and progression .................... 3
# Table of Contents

1.2 Proposed method .................................................. 4  
  1.2.1 Objectives .................................................... 4  
  1.2.2 Overview ..................................................... 5  
  1.2.3 Contributions ............................................... 6  
1.3 Thesis outline ..................................................... 7  

2 Background on Deep Learning ...................................... 11  
  2.1 Supervised learning .............................................. 12  
    2.1.1 Dense neural networks .................................... 13  
    2.1.2 Convolutional neural networks ........................... 15  
  2.2 Unsupervised learning .......................................... 17  
    2.2.1 From restricted Boltzmann machines to deep belief networks 18  
    2.2.2 Variants of restricted Boltzmann machines and deep belief networks 21  

3 Training of Convolutional Models in the Frequency Domain .......... 28  
  3.1 Related work ................................................... 28  
  3.2 Overview ....................................................... 30  
  3.3 Algorithm ...................................................... 30  
    3.3.1 Training in the spatial domain ........................... 30  
    3.3.2 Training in the frequency domain ....................... 32  
    3.3.3 Mapping of strided convolutions to stride-1 convolutions 35  
    3.3.4 GPU implementation and memory considerations ........... 36  
  3.4 Evaluation of running time .................................... 39  
    3.4.1 Comparison of running times for training sconvDBNs ........ 39  
    3.4.2 Comparison of running times for calculating convolutions with cuDNN 45  
  3.5 Conclusions ..................................................... 50
# 4 White Matter Lesion Segmentation

4.1 Introduction ........................................... 51
4.2 Related work ........................................... 51
    4.2.1 Unsupervised methods .......................... 52
    4.2.2 Supervised methods .............................. 53
    4.2.3 Patch-based deep learning methods .......... 53
    4.2.4 Fully convolutional methods ................. 54
4.3 Methods .................................................. 55
    4.3.1 Model architecture .............................. 57
    4.3.2 Gradient calculation ............................ 59
    4.3.3 Training .......................................... 62
    4.3.4 Implementation .................................... 64
4.4 Experiments and results .............................. 64
    4.4.1 Data sets and preprocessing .................... 65
    4.4.2 Comparison to other methods ................... 68
    4.4.3 Measures of segmentation accuracy .......... 69
    4.4.4 Training parameters ............................. 72
    4.4.5 Impact of the training set size on the segmentation performance ..... 73
    4.4.6 Comparison on public data sets ............... 74
    4.4.7 Comparison of network architectures, input modalities, and publicly available methods on clinical trial data ................................. 79
    4.4.8 Comparison for different lesion sizes ........ 82
4.5 Discussion .............................................. 85
## Table of Contents

5 **Manifold Learning by Deep Learning** ............................................. 88

  5.1 Introduction ................................................................. 88
  5.2 Manifold learning for medical image analysis .......................... 89
  5.3 Manifold learning using deep belief networks .......................... 92
    5.3.1 General architecture ................................................. 92
    5.3.2 Unit types .......................................................... 93
    5.3.3 Incorporating a region of interest .................................. 94
  5.4 Manifold of MRIs of Alzheimer’s disease patients ...................... 95
    5.4.1 Data sets and preprocessing ....................................... 96
    5.4.2 Method ............................................................. 96
    5.4.3 Results ............................................................. 97
  5.5 Variability of morphology and lesion distribution ..................... 101
    5.5.1 Data sets and preprocessing ....................................... 102
    5.5.2 Method ............................................................. 102
    5.5.3 Results ............................................................. 108
  5.6 Summary ................................................................. 112

6 **Conclusions and Future Work** ............................................... 113

  6.1 Contributions ............................................................. 114
  6.2 Future work ............................................................... 115
    6.2.1 New applications of deep learning ............................... 115
    6.2.2 Improving the performance on small data sets using data augmentation 116
    6.2.3 Rotation-invariant neural networks ............................... 117
    6.2.4 Increasing the expressive power of single neurons ............. 118

**Bibliography** ............................................................................. 120
List of Tables

2.1 Key variables and notation. For notational simplicity, we assume the input images to be square 2D images. ........................................... 23

3.1 Comparison of the memory required for storing key variables using different training methods ............................................................ 39

3.2 Training parameters. ....................................................................... 40

3.3 Hardware specification of our test system for training sconvDBNs using different implementations for calculating convolutions. .................. 40

3.4 CNN layer parameters for the comparison with cuDNN ..................... 46

3.5 Hardware specification of our test system for the comparison with cuDNN. 46

3.6 Comparison of running times for calculating key operations for training a CNN layer for different batch sizes .............................................. 48

3.7 Comparison of running times of time critical operations of two example deep learning models. .............................................................. 49

4.1 Population, disease and scanner characteristics of the two clinical trial data sets ........................................................................ 66

4.2 Parameters of the 3-layer CEN for the evaluation on the challenge data sets. 75

4.3 Comparison of our method with state-of-the-art lesion segmentation methods 77
4.4 Selected methods out of the 52 entries submitted for evaluation to the MICCAI 2008 MS lesion segmentation challenge 77
4.5 Comparison of our method with the second and third ranked methods from the ISBI MS lesion segmentation challenge 78
4.6 Parameters of the 3-layer CEN used on the clinical trial data set. 79
4.7 Parameters of the 7-layer CEN-s used on the clinical trial data set. 80
4.8 Comparison of the segmentation accuracy of different CEN models, other methods, and input modalities. 81
4.9 Lesion size groups as used for the detailed analysis. 82

5.1 Parameters of the DBN used to learn the manifold of brain MRIs of AD and healthy subjects. 97
5.2 Pearson correlation of demographic and clinical parameters with manifold coordinates 101
5.3 Training parameters of the morphology DBN, lesion DBN, and joint DBN 105
5.4 Pearson correlations of clinical scores with distribution parameters and imaging biomarkers 110
List of Figures

1.1 Demyelination in MS .................................................. 2

2.1 Schematic depiction for the calculations in a dense neural network .... 14
2.2 Convolutional neural network with two convolutional layers, one pooling layer and one dense layer ............................................ 16
2.3 Graph representation of an RBM with 3 hidden and 5 visible units .... 18

3.1 Comparison of training algorithms of convRBMs in the spatial and frequency domain ....................................................... 32
3.2 Mapping of strided convolutions to stride-1 convolutional ................. 36
3.3 Comparison of running times for training a sconvRBMs on 2D images ... 42
3.4 Comparison of running times for training a sconvRBMs on 3D images ... 44
3.5 Comparison of running times of key operations for training a single CNN layer. 47
3.6 Comparison of running times of key operations for training a single CNN layer. 48

4.1 Pre-training and fine-tuning a 7-layer convolutional encoder network with shortcut ............................................................... 56
4.2 Improvement in the mean DSC computed on the training and test sets during training ............................................................... 72
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>ROC curves for different sensitivity ratios $r$</td>
<td>73</td>
</tr>
<tr>
<td>4.4</td>
<td>Comparison of DSC scores calculated on the training and test sets for varying numbers of training samples</td>
<td>74</td>
</tr>
<tr>
<td>4.5</td>
<td>Example segmentations of our method for three different subjects from the MICCAI challenge data set</td>
<td>76</td>
</tr>
<tr>
<td>4.6</td>
<td>Impact of increasing the depth of the network on the segmentation performance of very large lesions</td>
<td>83</td>
</tr>
<tr>
<td>4.7</td>
<td>Comparison of segmentation performance of different CEN architectures for small lesions</td>
<td>83</td>
</tr>
<tr>
<td>4.8</td>
<td>Comparison of segmentation accuracy and lesion detection measures of a 7-layer CEN-s and LST-LGA for different lesion sizes</td>
<td>84</td>
</tr>
<tr>
<td>5.1</td>
<td>General architecture of the DBN used to learn the manifold of brain MRIs</td>
<td>94</td>
</tr>
<tr>
<td>5.2</td>
<td>Generalizability vs. specificity of a DBN for different numbers of layers</td>
<td>98</td>
</tr>
<tr>
<td>5.3</td>
<td>Axial slices from generated volumes from the manifold</td>
<td>99</td>
</tr>
<tr>
<td>5.4</td>
<td>Axial slices of volumes from the training set plotted against their manifold coordinates</td>
<td>100</td>
</tr>
<tr>
<td>5.5</td>
<td>Proposed model for discovering patterns of variability in MS brain</td>
<td>103</td>
</tr>
<tr>
<td>5.6</td>
<td>Slices from generated volumes from the morphology, lesion, and joint model</td>
<td>109</td>
</tr>
<tr>
<td>5.7</td>
<td>Axial and mid-sagittal slices of volumes representing the spectrum of MSFC scores</td>
<td>111</td>
</tr>
</tbody>
</table>
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9-HPT</td>
<td>9-hole peg test</td>
</tr>
<tr>
<td>AD</td>
<td>Alzheimer’s disease</td>
</tr>
<tr>
<td>ADNI</td>
<td>Alzheimer’s Disease Neuroimaging Initiative</td>
</tr>
<tr>
<td>CD</td>
<td>contrastive divergence</td>
</tr>
<tr>
<td>CEN</td>
<td>convolutional encoder network</td>
</tr>
<tr>
<td>CEN-s</td>
<td>convolutional encoder network with shortcut connections</td>
</tr>
<tr>
<td>CNN</td>
<td>convolutional neural network</td>
</tr>
<tr>
<td>CNS</td>
<td>central nervous system</td>
</tr>
<tr>
<td>convDBN</td>
<td>convolutional deep belief network</td>
</tr>
<tr>
<td>convRBM</td>
<td>convolutional restricted Boltzmann machine</td>
</tr>
<tr>
<td>cuDNN</td>
<td>NVIDIA CUDA Deep Neural Network library</td>
</tr>
<tr>
<td>DBN</td>
<td>deep belief network</td>
</tr>
<tr>
<td>DNN</td>
<td>dense neural network</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DSC</td>
<td>Dice similarity coefficient</td>
</tr>
<tr>
<td>EMS</td>
<td>expectation maximization segmentation</td>
</tr>
<tr>
<td>fCNN</td>
<td>fully convolutional neural network</td>
</tr>
<tr>
<td>FFT</td>
<td>fast Fourier transform</td>
</tr>
<tr>
<td>FLAIR</td>
<td>fluid-attenuated inversion recovery</td>
</tr>
<tr>
<td>FLIRT</td>
<td>FSL linear image registration tool</td>
</tr>
<tr>
<td>FN</td>
<td>false negative</td>
</tr>
<tr>
<td>FNIRT</td>
<td>FSL non-linear image registration tool</td>
</tr>
<tr>
<td>FP</td>
<td>false positive</td>
</tr>
<tr>
<td>GPU</td>
<td>graphics processing unit</td>
</tr>
<tr>
<td>ICBM</td>
<td>International Consortium for Brain Mapping</td>
</tr>
<tr>
<td>Isomap</td>
<td>isometric feature mapping</td>
</tr>
<tr>
<td>LEM</td>
<td>Laplacian eigenmaps</td>
</tr>
<tr>
<td>LFPR</td>
<td>lesion-wise false positive rate</td>
</tr>
<tr>
<td>LL</td>
<td>lesion load</td>
</tr>
<tr>
<td>LLE</td>
<td>locally linear embedding</td>
</tr>
<tr>
<td>LST</td>
<td>Lesion Segmentation Toolbox</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>LST-LGA</td>
<td>lesion growth algorithm</td>
</tr>
<tr>
<td>LST-LPA</td>
<td>lesion prediction algorithm</td>
</tr>
<tr>
<td>LTPR</td>
<td>lesion-wise true positive rate</td>
</tr>
<tr>
<td>MLE</td>
<td>maximum likelihood estimation</td>
</tr>
<tr>
<td>MMSE</td>
<td>mini-mental state examination</td>
</tr>
<tr>
<td>MNI</td>
<td>Montreal Neurological Institute</td>
</tr>
<tr>
<td>MOPS</td>
<td>model of population and subject</td>
</tr>
<tr>
<td>MR</td>
<td>magnetic resonance</td>
</tr>
<tr>
<td>MRF</td>
<td>Markov random field</td>
</tr>
<tr>
<td>MRI</td>
<td>magnetic resonance imaging</td>
</tr>
<tr>
<td>MS</td>
<td>multiple sclerosis</td>
</tr>
<tr>
<td>nBV</td>
<td>normalized brain volume</td>
</tr>
<tr>
<td>NReLU</td>
<td>noisy rectified linear unit</td>
</tr>
<tr>
<td>OASIS</td>
<td>Open Access Series of Imaging Studies</td>
</tr>
<tr>
<td>PASAT</td>
<td>paced auditory serial addition test</td>
</tr>
<tr>
<td>PCD</td>
<td>persistent CD</td>
</tr>
<tr>
<td>PDw</td>
<td>proton density-weighted</td>
</tr>
<tr>
<td>pp</td>
<td>percentage points</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>PPMS</td>
<td>primary progressive MS</td>
</tr>
<tr>
<td>PPV</td>
<td>positive predictive value</td>
</tr>
<tr>
<td>PRMS</td>
<td>progressive relapsing MS</td>
</tr>
<tr>
<td>RBM</td>
<td>restricted Boltzmann Machine</td>
</tr>
<tr>
<td>RF</td>
<td>random forest</td>
</tr>
<tr>
<td>RMSE</td>
<td>root mean squared error</td>
</tr>
<tr>
<td>ROC</td>
<td>receiver operating characteristic</td>
</tr>
<tr>
<td>ROI</td>
<td>region of interest</td>
</tr>
<tr>
<td>RRMS</td>
<td>relapsing remitting MS</td>
</tr>
<tr>
<td>sconvDBN</td>
<td>strided convolutional deep belief network</td>
</tr>
<tr>
<td>sconvRBM</td>
<td>strided convolutional restricted Boltzmann machine</td>
</tr>
<tr>
<td>SGD</td>
<td>stochastic gradient descent</td>
</tr>
<tr>
<td>SLS</td>
<td>Salem Lesion Segmentation</td>
</tr>
<tr>
<td>SPMS</td>
<td>secondary progressive MS</td>
</tr>
<tr>
<td>SSD</td>
<td>sum of squared differences</td>
</tr>
<tr>
<td>SVM</td>
<td>support vector machine</td>
</tr>
<tr>
<td>T1w</td>
<td>T1-weighted</td>
</tr>
<tr>
<td>T25W</td>
<td>timed 25-foot walk</td>
</tr>
<tr>
<td>T2w</td>
<td>T2-weighted</td>
</tr>
</tbody>
</table>
List of Abbreviations

TP  true positive
TPR true positive rate
VD  relative absolute volume difference
Acknowledgements

First and foremost, I want to express my deep gratitude to my supervisor Roger Tam, without whom I would not have started my PhD at UBC. From the very beginning of my internship at the MS/MRI research group until the end of my PhD, Roger has created a very kind, enjoyable, and productive working environment. He gave me the freedom and trust, which allowed me to pursue my personal research interests. I will miss our weekly meetings, which not only allowed me to discuss new ideas and potential struggles, but which were also filled with many joyful conversations. I’m most grateful for the unprecedented amount of time he dedicates to the supervision of his students, which has always motivated me to do the best research I can.

I am very grateful for the continued support and valuable feedback from my co-supervisor Rafeef Abugharbieh and all members of my supervisory committee: Purang Abolmaesumi, Jane Z. Wang, and Ghassan Hamarneh. I would also like to thank my departmental and university defense committee, Robert Rohling, Jim Little, and Martin McKeown, and the external examiner Daniel Rueckert for their valuable criticism and insightful feedback.

I’d like to thank everyone from the MS/MRI research group I’ve had the pleasure to be working with. Special thanks goes to Anthony Traboulsee and David K.B. Li for providing insightful feedback about potential clinical applications of my research, Andrew
Acknowledgements

Riddehough for providing a unique perspective about the operation of clinical trials, Kevin Lam and Rasmus Storjohann, who have taught me lots about software development, and Ken Bigelow and Vilmos Soti for maintaining a productive working environment.

I want to thank all my friends I’ve met at UBC who’ve made my everyday a joyful one. I want to thank my labmates Chunfang, Fahime, Saurabh, Youngjin, Robert, Lisa, Vanessa, and Niño for many stimulating discussions and welcome distractions, when I needed one. I also want to thank my friends from the other labs, Al, Antonio, Colin, Nishant and everyone from BMEGSA and ECEGSA for making me feel a part of the UBC/SFU research community.

Finally, I want to express my gratitude to my family, who have supported me throughout all these years. I want to thank my parents and my parents-in-law, for being understanding and supportive of our decision to move to Canada, even if it meant that we sometimes won’t see each other for more than a year. I also owe the deepest gratitude to my wife Anne, who has just been amazing since the moment I met her. Her encouragement and warming love has carried me through all the years and allowed me to find the right balance between focusing on my research and enjoying this beautiful country.
Dedication

— To my wonderful wife.
Chapter 1

Introduction

1.1 Motivation

1.1.1 A Short Introduction to Multiple Sclerosis

Multiple sclerosis (MS) is a chronic and degenerative disease of the central nervous system (CNS) characterized by the formation of inflammatory and demyelinating lesions. Presumably due to the breakdown of the blood-brain-barrier, the body’s own immune system attacks the myelin sheaths that act as insulating covers of the axons of neurons (see Figure 1.1). This disrupts the ability of parts of the nervous system to communicate, and can lead to the manifestation of a large range of different signs and symptoms. The brain is a very plastic organ and is often able to compensate for the damage by forming new neural pathways. In the later stages of the diseases, however, the amount of tissue damage is often too high to be offset, which leads to the progressive accumulation of disease.

The clinical presentation of MS is very heterogeneous due to the wide range of areas of the brain and spinal cord that can be affected. Characteristic but not specific physical signs and symptoms include the loss of sensitivity or changes in sensation such as tingling
Figure 1.1: Due to the break down of the blood brain barrier, the body’s own immune system attacks the myelin sheaths of the axons, which causes the formation of demyelinating lesions, visible primarily in the white matter on conventional magnetic resonance imaging (MRI) scans. Lesions are highlighted in white.

or numbness, which typically starts in the fingers and toes, as well as muscle weakness, difficulty in moving, difficulties with coordination and balance, problems with speech or swallowing, visual problems, feeling tired, acute or chronic pain, and bladder and bowel difficulties. In addition, MS often leads to cognitive problems such as having difficulties learning and remembering information, or psychiatric and emotional problems such as depression or frequent mood swings.

The course of the disease is unpredictable. Most patients (around 80%) are initially diagnosed as having relapsing remitting MS (RRMS), which is characterized by alternating periods of worsening due to inflammatory attacks and the formation of new lesions, and periods of remission and recovery. The majority (around 65%) of RRMS patients transition to secondary progressive MS (SPMS). In this stage, the body is no longer able
to compensate for the tissue damage, which leads to the unremitting and progressive accumulation of disability. Other types of MS include the primary progressive form (PPMS), characterized by the progression of disability from onset with no remissions after the initial symptoms, and progressive relapsing MS (PRMS), which shows progressive accumulation of disability in addition to clear superimposed relapses.

1.1.2 Measuring Disease State and Progression

There is currently no cure for MS. Existing therapies that focus on symptomatic management and prevention of further damage have variable degrees of effectiveness, although several recent breakthroughs are promising. To monitor and further our understanding of the disease, many biomarkers have been developed that allow for the objective measurement of normal and pathogenic processes, as well as the monitoring of treatment effect. Current MS biomarkers can be roughly classified into generic-immunogenetic, laboratorial, and imaging biomarkers (Katsavos and Anagnostouli, 2013). For this thesis, we focus on the accurate measurement of existing lesion-based imaging biomarkers such as lesion volume and lesion count, and the development of new biomarkers that capture changes in brain morphology and white matter lesions—two hallmarks of MS pathology.

Accurate segmentation of MS lesions has shown to be a very challenging due to the large variability in lesion size, shape, intensity, and locations, as well as the large variability in imaging contrasts produced by scanners used at different clinical sites. Despite a growing interest in developing fully automatic lesion segmentation methods, semi-automatic methods are still the standard in clinical research, although their use is time-consuming, laborious, and potentially biasing. It is therefore highly desirable to develop a fully-automatic lesion segmentation method that is robust to large variability, while still being able to segment lesions with high accuracy and sensitivity.
Imaging biomarkers used in clinical trials mostly focus on volumetric measures of global and local changes, which are important and relatively easy to compute, but only correlate modestly with clinical scores. One reason for the modest correlation is that they do not reflect potentially important structural variations, such as shape changes in the brain and the spatial dispersion of lesions. Therefore, it would be highly desirable to develop biomarkers that capture potentially important patterns of the variability in brain morphology and lesion distribution, which would advance our understanding of the complex pathology of MS.

1.2 Proposed Method

1.2.1 Objectives

The clinical motivation of the thesis is to develop methods that facilitate the automatic measurement of MS disease state and progression that are visible on conventional structural MRIs. To that end, we have identified two key applications. One is the development of a fully automatic lesion segmentation method that is able to segment lesions over a large range of sizes and in the presence of varying imaging contrasts and imaging artifacts produced by different scanners, which would allow for the accurate measurement of lesion-based imaging biomarkers such as lesion load and lesion count. The other key application is the development of a method that automatically discovers potentially important patterns of variability in brain morphology and lesion distribution, with the goal to derive new imaging biomarkers that correlate stronger with the clinical measures of MS disability than traditionally used volumetric measures. The global objective of the thesis is to determine the capabilities of deep learning (LeCun et al., 2015) for these two clinical applications.
1.2.2 Overview

The two methods developed for segmenting MS lesions and modelling patterns of variability are both based on deep learning, a field within machine learning that is inspired by the learning capabilities of the brain. The human brain has often served as a model for computer vision algorithms. For example, SIFT features (Lowe, 1999), inspired by neurons of the inferior temporal cortex, have proved to be very robust for object recognition. Resembling the receptive field of simple cells of the primary visual cortex, 2D Gabor filters have been used to describe textures for segmentation (Grigorescu et al., 2002). In contrast, deep learning algorithms try to mimic how the visual system learns instead of copying what it has learned. First evidence for the learning capabilities of the visual system were found by (Wiesel and Hubel, 1963), who investigated the visual cortex of cats. They showed that the receptive fields of neurons are learned from a continuous stream of images early in the development of the visual system (Wiesel and Hubel, 1963), but they are also fine-tuned later (Karni and Sagi, 1991). This allows the neurons of the visual cortex to adapt to the type of images to which it is exposed. While it is difficult, for example, for an average person to recognize the differences between cows of the same breed, the feature detecting neurons of the visual system of cattle farmers are highly tuned to their appearance, allowing them to recognize individual cows easily. This suggests that a learning-based model for classification should not only learn to perform the requested task based on a set of pre-defined features, but also learn the features that are most suitable to perform the task. The joint learning of feature extraction and prediction, also known as end-to-end learning, is possible through the use of deep learning methods, which use multiple layers of nonlinear processing units to learn a feature hierarchy directly from the input data without a dedicated feature extraction step.
Deep learning has successfully been used in many research areas such as object recognition (Krizhevsky et al., 2012), speech recognition (Hinton et al., 2012), natural language understanding (Collobert et al., 2011b), and language translation (Sutskever et al., 2014). Deep learning methods are particularly successful due to their ability to recognize complex and highly nonlinear patterns in large amounts of training data, which facilitates the learning of models that are robust to large variability. This motivates the use of deep learning methods for segmenting MS lesions, because the large variability in lesion shape, size, contrast, and location as well as changes in imaging contrasts produced by different MRI scanners make lesion segmentation challenging. Beyond voxel classification, deep learning methods can also be used to model highly nonlinear patterns of variability in groups of images. This allows deep learning models such as deep belief networks to be used for manifold learning, e.g., of hand written digits (Hinton et al., 2006) or, as we will show in Chapter 5, brain MRIs (Brosch and Tam, 2013). However, deep learning algorithms as implemented by widely used deep learning frameworks were originally developed for application to small 2D images and do not scale well to large 3D volumes in terms of training time and memory requirements, which prevents the use of out-of-the-box implementations for 3D medical image analysis.

1.2.3 Contributions

In the course of developing deep learning-based methods for MS lesion segmentation and pattern discovery, we have made the following main contributions:

1. We have developed a novel training algorithm for convolutional deep belief networks and convolutional neural networks that performs training in the frequency domain. The speed-ups gained by our method compared to state-of-the-art spatial domain
implementations and the reduced memory requirements compared to other frequency domain methods enable the application of deep learning to high-resolution 3D medical images.

2. We have developed a neural network architecture that jointly learns features at different scales that are tuned to segmenting MS lesions and performs the segmentation based on the automatically learned features. The joint learning of feature extractor and classifier facilitates the learning of features that are robust to the large variability of MS lesions and varying contrasts produced by different scanners.

3. We have developed a novel objective function for training neural networks that is suitable for the classification of vastly unbalanced classes, such as the segmentation of MS lesions, which typically comprise less than one percent of the image.

4. This is the first work to demonstrate that deep learning can be applied to manifold learning of brain MRIs.

5. We have developed a framework for modelling changes in brain morphology and lesion distribution with only a few parameters, which also show improved correlation with clinical scores compared to established volumetric imaging biomarkers.

1.3 Thesis Outline

The rest of this thesis is organized into five chapters as outlined below:

Chapter 2—Background on Deep Learning

In this chapter, we will briefly introduce the supervised and unsupervised deep learning models that form the basis for the lesion segmentation and manifold learning methods,
which are discussed further in Chapter 4 and Chapter 5. We will start with a description of dense neural networks (Farley and Clark, 1954; Rumelhart et al., 1986; Werbos, 1974) and convolutional neural networks (Fukushima, 1980; LeCun et al., 1989, 1998). In the second part, we will give a brief overview of restricted Boltzmann Machines (Freund and Haussler, 1992; Hinton, 2010), which are the building blocks of deep belief networks (Hinton et al., 2006).

**Chapter 3—Training of Convolutional Models in the Frequency Domain**

Deep learning has traditionally been computationally expensive and advances in training methods have been the prerequisite for improving its efficiency in order to expand its application to a variety of image classification problems. In this chapter, we address the problem of efficient training of convolutional deep belief networks by learning the weights in the frequency domain, which eliminates the time-consuming calculation of convolutions. An essential consideration in the design of the algorithm is to minimize the number of transformations to and from frequency space. We have evaluated the running time improvements using two standard benchmark data sets, showing a speed-up of up to 8 times on 2D images and up to 200 times on 3D volumes. In addition, we have directly compared the time required to calculate convolutions using our method with the NVIDIA CUDA Deep Neural Network library (cuDNN), the current state-of-the-art library for calculating 2D and 3D convolutions, with the results showing that our method can calculate convolutions up to 20 times faster than cuDNN. Our training algorithm makes training of convolutional deep belief networks and convolutional neural networks on 3D volumes with a resolution of up to $128 \times 128 \times 128$ voxels practical, which opens new directions for using deep learning for medical image analysis.
Chapter 4—White Matter Lesion Segmentation

In this chapter, we present a novel segmentation approach based on deep 3D convolutional encoder networks with shortcut connections and apply it to the segmentation of MS lesions in magnetic resonance images. Our model is a neural network that consists of two interconnected pathways, a convolutional pathway, which learns increasingly more abstract and higher-level image features, and a deconvolutional pathway, which predicts the final segmentation at the voxel level. The joint training of the feature extraction and prediction pathways allows for the automatic learning of features at different scales that are optimized for accuracy for any given combination of image types and segmentation task. In addition, shortcut connections between the two pathways allow high- and low-level features to be integrated, which enables the segmentation of lesions across a wide range of sizes. We have evaluated our method on two publicly available data sets (MICCAI 2008 and ISBI 2015 challenges) with the results showing that our method performs comparably to the top-ranked state-of-the-art methods, even when only relatively small data sets are available for training. In addition, we have compared our method with five freely available and widely used MS lesion segmentation methods (EMS, LST-LPA, LST-LGA, Lesion-TOADS, and SLS) on a large data set from an MS clinical trial. The results show that our method consistently outperforms these other methods across a wide range of lesion sizes.

Chapter 5—Manifold Learning by Deep Learning

Manifold learning of medical images plays a potentially important role for modelling anatomical variability within a population with applications that include segmentation, registration, and prediction of clinical parameters. In this chapter, we describe a novel
method for learning the manifold of 3D brain images and for building a statistical model of brain images that can automatically discover spatial patterns of variability in brain morphology and lesion distribution. We propose building such a model using a deep belief network, a layered network whose parameters can be learned from training images. In contrast to other manifold learning algorithms, this approach does not require a prebuilt proximity graph, which is particularly advantageous for modelling lesions, because their sparse and random nature makes defining a suitable distance measure between lesion images challenging. Our results show that this model can automatically learn a low-dimensional manifold of brain volumes that detects modes of variations that correlate to demographic and disease parameters. Furthermore, our model can automatically discover the classic patterns of MS pathology, as well as more subtle ones, and the computed parameters have strong relationships to MS clinical scores.

Chapter 6—Conclusions and Future Work

This chapter concludes the thesis with a brief summary of the problems addressed and key results. In addition, some directions for future work are given in the broader context of deep learning for medical imaging. In particular, we will give suggestions for new medical image analysis applications of deep learning and how to deal with relatively small data sets using data augmentation. In addition, we will discuss two potential advancements of neural networks that we believe will be particularly important for medical applications.
Chapter 2

Background on Deep Learning

Deep learning (LeCun et al., 2015) is a field within machine learning that has been studied since the early 1980s (Fukushima, 1980). However, deep learning methods did not gain in popularity until the late 2000s with the advent of fast general purpose graphics processors (Raina et al., 2009), layerwise pre-training methods (Hinton et al., 2006; Hinton and Salakhutdinov, 2006), and large data sets (Deng et al., 2009; Krizhevsky et al., 2012). Since then, deep learning methods have become the state of the art in many non-medical (Krizhevsky et al., 2012; Sainath et al., 2013) and medical (Ciresan et al., 2012; Kamnitsas et al., 2015) applications. There are many different algorithms and models that are commonly referred to as deep learning methods, all of which have two properties in common: 1) the use of multiple layers of nonlinear processing units for extracting features, and 2) the layers are organized to form a hierarchy of low-level to high-level features. Representing data in a feature hierarchy has many advantages for classification and other applications. To give an example of a feature hierarchy, let us consider the domain of face images. The lowest layer of the feature hierarchy is composed of the raw pixel intensities, which are the most basic features of an image. Multiple pixels
can be grouped to form general image features like edges and corners, which can be further combined to form face parts such as different variations of noses, eyes, mouths, and ears. Finally, multiple face parts can be combined to form a variety of face images. Learning a feature hierarchy facilitates the parameterization of a large feature space with a small number of values by capturing complex relationships between feature layers. For example, a feature hierarchy consisting of three prototypical shapes for mouths, eyes, ears, and noses is able to represent \(3 \times 3 \times 3 \times 3 = 81\) different prototypical faces with only \(3 + 3 + 3 + 3 = 12\) features. Without a hierarchical representation of the data, a model would require 81 prototypical face features to span the same face manifold.

In this chapter, we will briefly introduce the supervised and unsupervised deep learning models that form the basis for the lesion segmentation and manifold learning methods, which are discussed further in Chapter 4 and Chapter 5. We will start with a description of dense neural networks (DNNs) (Farley and Clark, 1954; Rumelhart et al., 1986; Werbos, 1974) and convolutional neural networks (CNNs) (Fukushima, 1980; LeCun et al., 1989, 1998). In the second part, we will give a brief overview of restricted Boltzmann Machines (RBMs) (Freund and Haussler, 1992; Hinton, 2010), which are the building blocks of deep belief networks (DBNs) (Hinton et al., 2006).

### 2.1 Supervised Learning

A typical pipeline for classifying images consists of two main steps. In the first step, predefined or learned features are extracted from the input images, which are then used to train a separate supervised learning model, such as a random forest (RF) (Breiman, 2001) or a support vector machine (SVM) (Cortes and Vapnik, 1995), to perform classification or prediction. Alternatively, classification and prediction can be performed with a single
model that takes the raw input data and produces the desired output, such as class probabilities. This type of learning is called end-to-end learning and has shown great potential for medical image analysis (Ciresan et al., 2012). The most popular models for end-to-end learning are neural networks due to their ability to learn a hierarchical set of features from raw input data. This allows the learning of features that are tuned for a given combination of input modalities and classification task, but is more prone to overfitting than unsupervised feature learning methods, especially when the amount of labeled data is limited. In this section, we will start with an introduction to dense neural networks, followed by a concise overview of convolutional neural networks.

### 2.1.1 Dense Neural Networks

A dense neural network (see Figure 2.1) is a deterministic function that maps input data to the desired outputs through the successive application of multiple nonlinear mappings of the following form

\[
\mathbf{z}_l = \mathbf{W}_l \mathbf{x}_{l-1} + \mathbf{b}_l, \quad \text{(2.1)}
\]

\[
\mathbf{x}_l = f_l(\mathbf{z}_l), \quad \text{(2.2)}
\]

where \( \mathbf{x}_l \) denotes a vector containing the units of layer \( l \), \( \mathbf{x}_0 \) denotes a vector containing the input of the neural network, \( \mathbf{x}_L \) denotes a vector containing the output, \( L \) is the number of computational layers, \( f_l \) are transfer functions, \( \mathbf{W}_l \) are weight matrices, and \( \mathbf{b}_l \) are bias terms. Popular choices for the transfer function are the sigmoid function \( f(x) = \text{sigm}(x) \) and the rectified linear function \( f(x) = \max(0, x) \). The same transfer function is typically used for all layers except for the output layer. The choice of the output transfer function
depends on the learning task. For classification, a 1-of-\( n \) encoding of the output class is usually used in combination with the softmax transfer function defined as

\[
\text{softmax}(a)_i = \frac{\exp(a_i)}{\sum_{j=1}^{n} \exp(a_j)},
\]

where \( a \) denotes an \( n \)-dimensional output vector.

Given a training set \( D = \{(x_0^{(i)}, y^{(i)}) \mid i \in [1, N]\} \), a neural network is trained by minimizing the error between the predicted outputs \( x_L^{(i)} \) and the given labels \( y^{(i)} \)

\[
\hat{\theta} = \arg \min_{\theta} \sum_{i=1}^{N} E(x_L^{(i)}, y^{(i)}),
\]

where \( \theta \) denotes the trainable parameters of the neural network. Typical choices for the error function are the sum of squared differences (SSD) and the cross-entropy. The minimization problem can be solved using stochastic gradient descent (SGD) (Polyak and Juditsky, 1992; Rumelhart et al., 1986), which requires the calculation of the gradient of
the error function with respect to the model parameters. The gradient can be calculated
by backpropagation (Werbos, 1974) as follows

$$
\delta_L = \nabla_{x_L} E \cdot f'_L(z_L),
$$

(2.5)

$$
\delta_l = (W_{l+1}^T \delta_{l+1}) \cdot f'_l(z_l)
$$

for \( l < L \),

(2.6)

$$
\nabla_{W_l} E = \delta_l x_{l-1}^T
$$

(2.7)

$$
\nabla_{b_l} E = \delta_l
$$

(2.8)

where \( \nabla_{x_L} E \) denotes the gradient of the error function with respect to the predicted output
and \( \cdot \) denotes element-wise multiplication.

### 2.1.2 Convolutional Neural Networks

The structure of CNNs is inspired by the complex arrangement of simple and complex cells
found in the visual cortex (Hubel and Wiesel, 1962, 1968). Simple cells are only connected
to a small sub-region of the previous layer and need to be tiled to cover the entire visual
field. In a CNN (see Figure 2.2), simple cells are represented by convolutional layers,
which exhibit a similar mechanism of local connectivity and weight sharing. Complex
cells combine the activations of simple cells to add robustness to small translations. These
cells are represented in the form of pooling layers. After several alternating convolutional
and pooling layers, the activations of the last convolutional layer are fed into one or more
dense layers to carry out the final classification.

For multimodal 3D volumes, the neurons of convolutional and pooling layers are
arranged in a 4D array, where the first three dimensions correspond to the dimensions of
Figure 2.2: Convolutional neural network with two convolutional layers, one pooling layer and one dense layer. The activations of the last layer are the output of the network.

the input volume, and the forth dimension indexes the input modality or channel. The activations of the output of a convolutional layer are calculated by

$$x^{(l)}_j = f \left( \sum_{i=1}^{C} \tilde{w}^{(l)}_{ij} * x^{(l-1)}_i + b^{(l)}_j \right),$$  \hspace{1cm} (2.9)

where $l$ is the index of a convolutional layer, $x^{(l)}_j$ denotes the $j$th channel of the output volume, $w^{(l)}_{ij}$ is a 3D filter kernel connecting the $i$th channel of the input volume to the $j$th channel of the output volume, $b^{(l)}_j$ denotes the bias term of the $j$th output channel, and $\tilde{w}$ denotes a flipped version of $w$, i.e. $\tilde{w}(a) = w(-a)$. CNNs can be trained using stochastic gradient descent, where the gradient can be derived analogously to dense neural networks and calculated using backpropagation (LeCun et al., 1989, 1998).

Different types of operations (Scherer et al., 2010) have been proposed for the pooling layers, with the common goal of creating a more compact representation of the input data.
The most commonly used type of pooling is max-pooling. Therefore, the input to the pooling layer is divided into small blocks and only the maximum value of each block as passed on to the next layer, which makes the representation of the input invariant to small translations in addition to reducing its dimensionality.

A major challenge for gradient-based optimization methods is the choice of an appropriate learning rate. Classic stochastic gradient descent (LeCun et al., 1998) uses a fixed or decaying learning rate, which is the same for all parameters of the model. However, the partial derivatives of parameters of different layers can vary substantially in magnitude, which can require different learning rates. In recent years, there has been an increasing interest in developing methods for automatically choosing independent learning rates. Most methods (e.g., AdaGrad by Duchi et al., 2011; AdaDelta by Zeiler, 2012; RMSprop by Dauphin et al., 2015; and Adam by Kingma and Ba, 2014) collect different statistics of the partial derivatives over multiple iterations and use this information to set an adaptive learning rate for each parameter. This is especially important for the training of deep networks, where the optimal learning rates often differ greatly for each layer.

2.2 Unsupervised Learning

One of the most important applications of deep learning is to learn patterns of variability in the form of a feature hierarchy from unlabeled images. The key to learning such a hierarchy is the ability of deep models to be trained layer by layer, where each layer acts as a nonlinear feature extractor. Various methods have been proposed for feature extraction from unlabeled images. In this section, we will first introduce the restricted Boltzmann machines (Freund and Haussler, 1992; Hinton, 2010), which are the building blocks of the later described deep belief networks (Hinton et al., 2006).
2.2.1 From Restricted Boltzmann Machines to Deep Belief Networks

An RBM is a probabilistic graphical model defined by a bipartite graph as shown in Figure 2.3. The units of the RBM are divided into two layers, one of visible units $v$ and the other of hidden units $h$. There are no direct connections between units within either layer. An RBM defines the joint probability of visible and hidden units in terms of the energy $E$,

$$p(v, h \mid \theta) = \frac{1}{Z(\theta)} e^{-E(v, h \mid \theta)}.$$  \hspace{1cm} (2.10)

When the visible and hidden units are binary, the energy is defined as

$$-E(v, h \mid \theta) = \sum_{i,j} v_i w_{ij} h_j + \sum_i b_i v_i + \sum_j c_j h_j,$$  \hspace{1cm} (2.11)

$$= v^T W h + b^T v + c^T h,$$  \hspace{1cm} (2.12)

where $Z(\theta)$ is a normalization constant, $W$ denotes the weight matrix that connects the visible units with the hidden units, $b$ is a vector containing the visible bias terms, $c$ is a vector containing the hidden bias terms, and $\theta = \{W, b, c\}$ are the trainable parameters of the RBM.
Inference

The hidden units represent patterns of similarity that can be observed in groups of images. Given a set of model parameters $\theta$, the features of an image can be extracted by calculating the expectation of the hidden units. The posterior distribution of the hidden units given the visible units can be calculated by

$$p(h_j = 1 \mid v, \theta) = \text{sigm}(w_{ij}^T v + c_j),$$

(2.13)

where $w_{ij}$ denotes the $j$th column vector of $W$, and $\text{sigm}(x)$ is the sigmoid function defined as $\text{sigm}(x) = (1 + \exp(-x))^{-1}, x \in \mathbb{R}$. An RBM is a generative model, which allows for the reconstruction of an input signal given its features. This is achieved by calculating the expectation of the visible units given the hidden units. The posterior distribution $p(v_i = 1 \mid h, \theta)$ can be calculated by

$$p(v_i = 1 \mid h, \theta) = \text{sigm}(w_{i,j}^T h + b_i),$$

(2.14)

where $w_{i,j}$ denotes the $i$th row vector of $W$. Reconstructing the visible units can be used to visualize the learned features. To visualize the features associated with a particular hidden unit, all other hidden units are set to zero and the expectation of the visible units is calculated, which represents the pattern that causes a particular hidden unit to be activated.

Training

RBMs can be trained by maximizing the likelihood or, more commonly, the log-likelihood of the training data, $D = \{v_n \mid n \in [1, N]\}$, which is called maximum likelihood estimation.
(MLE). The gradient of the log-likelihood function with respect to the weights, $W$, is given by the mean difference of two expectations

$$\nabla W \log p(D | \theta) = \frac{1}{N} \sum_{n=1}^{N} E[\mathbf{v}\mathbf{h}^T | \mathbf{v}_n, \theta] - E[\mathbf{v}\mathbf{h}^T | \theta].$$

(2.15)

The first expectation can be estimated using a mean field approximation

$$E[\mathbf{v}\mathbf{h}^T | \mathbf{v}_n, \theta] \approx E[\mathbf{v} | \mathbf{v}_n, \theta]E[\mathbf{h}^T | \mathbf{v}_n, \theta],$$

(2.16)

$$= \mathbf{v}_n E[\mathbf{h}^T | \mathbf{v}_n, \theta].$$

(2.17)

The second expectation is typically estimated using a Monte Carlo approximation

$$E[\mathbf{v}\mathbf{h}^T | \theta] \approx \frac{1}{S} \sum_{s=1}^{S} \mathbf{v}_s \mathbf{h}_s^T,$$

(2.18)

where $S$ is the number of generated samples, and $\mathbf{v}_s$ and $\mathbf{h}_s$ are samples drawn from $p(\mathbf{v} | \theta)$ and $p(\mathbf{h} | \theta)$, respectively. Samples from an RBM can be generated efficiently using block Gibbs sampling, in which the visible and hidden units are initialized with random values and alternately sampled given the previous state using

$$h_j = I(y_j < p(h_j = 1 | \mathbf{v}, \theta)) \quad \text{with } y_j \sim U(0, 1)$$

(2.19)

$$v_i = I(x_i < p(v_i = 1 | \mathbf{h}, \theta)) \quad \text{with } x_i \sim U(0, 1)$$

(2.20)

where $z \sim U(0, 1)$ denotes a sample drawn from the uniform distribution in the interval $[0, 1]$ and $I$ is the indicator function, which is defined as 1 if the argument is true and 0 otherwise. After several iterations, a sample generated by the Gibbs chain is distributed according to $p(\mathbf{v}, \mathbf{h} | \theta)$. 

20
If the Gibbs sampler is initialized at a data point from the training set and only one Monte Carlo sample is used to approximate the second expectation in (2.15), the learning algorithm is called contrastive divergence (CD) (Hinton, 2002). Alternatively, persistent CD (PCD) (Tieleman, 2008) uses several separate Gibbs chains to generate data independent samples from the model, which results in a better approximation of the gradient of the log-likelihood than CD.

To speed up the training of RBMs using either CD and PCD, the data set is usually divided into small subsets called mini-batches and a gradient step is performed for each mini-batch. To avoid confusion with a gradient step, the term “iteration” is generally avoided and the term “epoch” is used instead to indicate a sweep through the entire data set. Additional tricks to monitor and speed up the training of an RBM can be found in Hinton’s RBM training guide (Hinton, 2010). A detailed explanation and comparison of different training algorithms is given in Chapter 3.

**Deep Belief Networks**

A single RBM can be regarded as a nonlinear feature extractor. To learn a hierarchical set of features, multiple RBMs are stacked and trained layer by layer, where the first RBM is trained on the input data and subsequent RBMs are trained on the hidden unit activations computed from the previous RBMs. The stacking of RBMs can be repeated to initialize DBNs of any depth.

**2.2.2 Variants of Restricted Boltzmann Machines and Deep Belief Networks**

Many variants of RBMs and DBNs have been proposed to adapt them for different domains. In this section, we will first introduce the convolutional deep belief networks
(convDBNs), which allow DBNs to be applied to high-resolution images, followed by a discussion of different unit types, which allow DBNs to be applied to real-valued data like the intensities of some medical images.

**Convolutional Deep Belief Networks**

A potential drawback of DBNs is that the learned features are location dependent. Hence, features that can occur at many different locations in an image, such as edges and corners, must be relearned for every possible location, which dramatically increases the number of features required to capture the content of large images. To increase the translational invariance of the learned features, Lee et al. introduced the convDBN (Lee et al., 2009, 2011). In a convDBN, the units of each layer are organized in a multidimensional array that reflects the arrangement of pixels in the input images. The units of one layer are only connected to the units of a sub-region of the previous layer, and share the same weights with all other units of the same layer. This greatly reduces the number of trainable weights, which reduces the risk of overfitting, reduces the memory required to store the model parameters, speeds up the training, and thereby facilitates the application to high-resolution images.

A convDBN consists of alternating convolutional and pooling layers, which are followed by one or more dense layers. Each convolutional layer of the model can be trained in a
Table 2.1: Key variables and notation. For notational simplicity, we assume the input images to be square 2D images.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v^{(i)} )</td>
<td>a 2D array containing the units of the ( i )th input channel</td>
</tr>
<tr>
<td>( h^{(j)} )</td>
<td>a 2D array containing the units of ( j )th output channel or feature map</td>
</tr>
<tr>
<td>( w^{(ij)} )</td>
<td>a 2D array containing the weights of filter kernels connecting visible units ( v^{(i)} ) to hidden units ( h^{(j)} )</td>
</tr>
<tr>
<td>( b_i )</td>
<td>bias terms of the visible units</td>
</tr>
<tr>
<td>( c_j )</td>
<td>bias terms of the hidden units</td>
</tr>
<tr>
<td>( N_c )</td>
<td>number of channels of the visible units</td>
</tr>
<tr>
<td>( N_v )</td>
<td>width and height of the image representing the visible units</td>
</tr>
<tr>
<td>( N_k )</td>
<td>number of filters and feature maps</td>
</tr>
<tr>
<td>( N_h )</td>
<td>width and height of a feature map</td>
</tr>
<tr>
<td>( \cdot )</td>
<td>element-wise product followed by summation</td>
</tr>
<tr>
<td>( * )</td>
<td>valid convolution</td>
</tr>
<tr>
<td>( @ )</td>
<td>full convolution</td>
</tr>
<tr>
<td>( \tilde{w}^{(ij)} )</td>
<td>horizontally and vertically flipped version of ( w^{(ij)} ), i.e., ( \tilde{w}^{(ij)}<em>{uv} = w^{(ij)}</em>{N_w-u+1,N_w-v+1} ), where ( N_w ) denotes the width and height of a filter kernel</td>
</tr>
</tbody>
</table>

Greedy layerwise fashion by treating it as a convolutional restricted Boltzmann machine (convRBM). The energy of a convRBM with binary visible and hidden units is defined as

\[
E(v, h) = - \sum_{i=1}^{N_c} \sum_{j=1}^{N_k} \sum_{x,y=1}^{N_h} \sum_{u,v=1}^{N_w} h^{(j)}_{xy} w^{(ij)}_{uv} v^{(i)}_{x+u-1,y+v-1} - \sum_{i=1}^{N_c} b_i \sum_{x,y=1}^{N_v} v^{(i)}_{xy} - \sum_{j=1}^{N_k} c_j \sum_{x,y=1}^{N_h} h^{(j)}_{xy} \tag{2.21}
\]

\[
E(v, h) = - \sum_{i=1}^{N_c} \sum_{j=1}^{N_k} h^{(j)}_{ij} \cdot (\tilde{w}^{(ij)} * v^{(i)}) - \sum_{i=1}^{N_c} b_i \sum_{x,y=1}^{N_v} v^{(i)}_{xy} - \sum_{j=1}^{N_k} c_j \sum_{x,y=1}^{N_h} h^{(j)}_{xy} \tag{2.22}
\]

The key terms and notations are defined in Table 2.1. At the first layer, the number of channels \( N_c \) is one when trained on unimodal images, or equal to the number of input modalities when trained on multimodal images. For subsequent layers, \( N_c \) is equal to the number of filters of the previous layer.
The posterior distributions $p(h | v)$ and $p(v | h)$ can be derived from the energy equation and are given by

\[
p(h_{xy} = 1 | v) = \text{sigm} \left( \sum_{i=1}^{N_h} (\tilde{w}^{(ij)} * v^{(i)})_{xy} + c_j \right), \quad \text{and} \quad (2.23)
\]

\[
p(v_{xy} = 1 | h) = \text{sigm} \left( \sum_{j=1}^{N_v} (w^{(ij)} \odot h^{(j)})_{xy} + b_i \right). \quad (2.24)
\]

To train a convRBM on a set of images $D = \{v_n | n \in [1, N]\}$, the weights and bias terms can be learned by CD. During each iteration of the algorithm, the gradient of each parameter is estimated and a gradient step with a fixed learning rate is applied. The gradient of the filter weights can be approximated by

\[
\Delta w^{(ij)} \approx \frac{1}{N} (v_n^{(i)} * \tilde{h}_n^{(j)} - v_n'^{(i)} * \tilde{h}_n'^{(j)}), \quad (2.25)
\]

where $h_n^{(j)}$ and $h_n'^{(j)}$ are samples drawn from $p(h^{(j)} | v_n)$ and $p(h^{(j)} | v_n')$, and $v_n'^{(i)} = E[v^{(i)} | h_n]$.

**Strided Convolutional Models**

Strided convolutions are a type of convolution that shifts the filter kernel as a sliding window with a step size or stride $s > 1$, stopping at only $N_v/s$ positions. Replacing stride-1 convolutions with strided convolutions, we can define the energy function of a strided convolutional restricted Boltzmann machine (sconvRBM) as follows

\[
E(v, h) = - \sum_{i=1}^{N_v} \sum_{j=1}^{N_h} \sum_{x,y=1}^{N_h} \sum_{u,v=1}^{N_v} h^{(j)}_{xy} w^{(ij)}_{uv} v_{s(x-1)+u,s(y-1)+v} - \sum_{i=1}^{N_v} b_i \sum_{x,y=1}^{N_h} v^{(i)}_{xy} - \sum_{j=1}^{N_h} c_j \sum_{x,y=1}^{N_h} h^{(j)}_{xy}.
\quad (2.26)
\]
Strided convolutional RBMs have several advantages over traditional convRBMs. The use of strided convolutions reduces the number of hidden units per channel to $N_h = N_v / s$, hence significantly reducing training time and memory required for storing the hidden units during training. Furthermore, strided convolutional deep belief networks (sconvDBNs) do not require pooling layers to reduce the dimensionality, because dimensionality reduction is already performed within the sconvRBM layers. Consequently, inference in an sconvDBN is invertible, which allows for the visualization of detected patterns similar to DBNs. Rules for inference, sampling, and training of an sconvRBM can be derived analogous to convRBMs. Furthermore, in Section 3.3.3, we will show how to convert an sconvRBM into an equivalent convRBM by reorganizing the hidden units, which enables efficient training and inference of sconvRBMs in the frequency domain.

**Alternative Unit Types**

To model real-valued inputs like the intensities of some medical images, the binary visible units of an RBM can be replaced with Gaussian visible units, which leads to the following energy function

$$E(v, h | \theta) = \sum_{ij} \frac{v_i}{\sigma_i} w_{ij} h_j + \sum_i \frac{(v_i - b_i)^2}{2\sigma_i^2} + \sum_j c_j h_j,$$

where the mean of the $i$th visible unit is encoded in the bias term $b_i$, and its standard deviation is given by $\sigma_i$. Although approaches have been proposed for learning the standard deviation (Cho et al., 2011), the training data is often simply standardized to
have zero mean and unit variance, which yields the following simplified rules for inferring of the visible and hidden units:

\[
E[h_j | v, \theta] = \text{sigm}(w_{i,j}^T v + c_j), \tag{2.28}
\]

\[
E[v_i | h, \theta] = w_{i,j}^T h + b_i. \tag{2.29}
\]

A binary hidden unit can only encode two states. In order to increase the expressive power of the hidden units, Nair et al. proposed using noisy rectified linear units (NReLUs) (Nair and Hinton, 2010) as the hidden units, and showed that this can improve the learning performance of RBMs. The signal of an NReLU is the sum of an infinite number of binary units, all of which having the same weights but different bias terms. In the special case where the offsets of their bias terms are set to \(-0.5, -1.5, \ldots\), the sum of their probabilities and therefore the expectation of an NReLU is extremely close to having a closed form:

\[
E[h_j | v, \theta] = \sum_{i=1}^{\infty} \text{sigm}(w_{i,j}^T v + c_j - i + 0.5), \tag{2.30}
\]

\[
\approx \log(1 + \exp(w_{i,j}^T v + c_j)). \tag{2.31}
\]

However, sampling of this type of unit involves the repeated calculation of the sigmoid function, which can be time-consuming. If a sample is not constrained to being an integer, a fast approximation can be calculated with

\[
h_j \sim \max(0, \mu_j + \mathcal{N}(0, \text{sigm}(\mu_j))), \tag{2.32}
\]

\[
\mu_j = w_{i,j}^T v + c_j. \tag{2.33}
\]
where $\mathcal{N}(0, \sigma^2)$ denotes Gaussian noise.

In this section, we have introduced the most basic deep learning methods, which form the basis for the segmentation and manifold learning methods explained in Chapter 4 and Chapter 5, respectively. The next chapter details our training algorithm of convDBNs and CNNs in full, along with a comparison of alternative training methods.
Chapter 3

Training of Convolutional Models in the Frequency Domain

3.1 Related Work

Deep learning (LeCun et al., 2015) has traditionally been computationally expensive and advances in training methods have been the prerequisite for expanding its application to a variety of image classification problems. The development of layer-wise training methods (Hinton et al., 2006) greatly improved the efficiency of the training of deep belief networks (DBNs) (Hinton et al., 2006), which has made feasible the use of large sets of small images (e.g. 28 × 28), such as those used for hand-written digit recognition. Subsequently, new directions for speeding up the training of deep models were opened with the advance of programmable graphics cards (GPUs), which can perform thousands of operations in parallel. Raina et al. (2009) demonstrated that by using graphics cards, training of restricted Boltzmann Machines (RBMs) (Freund and Haussler, 1992; Hinton, 2010) on small image patches (e.g. 24 × 24) can be performed up to 70 times faster than
on the CPU, facilitating the application to larger training sets. However, the number of trainable weights of a deep belief network (DBN) increases greatly with the resolution of the training images, which can make training on large images impracticable. In order to scale DBNs to high-resolution images, Lee et al. (2009, 2011) introduced the convolutional deep belief network (convDBN), a deep generative model that uses weight sharing and local connectivity to reduce the number of trainable weights. They showed that a convDBN can be used to classify images with a resolution up to $200 \times 200$ pixels. To speed up the training of convolutional neural networks (CNNs) (Fukushima, 1980; LeCun et al., 1989, 1998) on high-resolution images, Krizhevsky et al. (2012) replaced traditional convolutions of the first layer of their CNN with strided convolutions, a type of convolution that shifts the filter kernel as a sliding window with a fixed step size or stride greater than one. Through the use of strided convolutions, the number of hidden units in each convolutional layer is greatly reduced, which reduces both training time and required memory. Using a highly optimized GPU implementation of convolutions, they were able to train a CNN on images with a resolution of $256 \times 256$ pixels, achieving state-of-the-art performance on the ILSVRC-2010 and ILSVRC-2012 natural image classification competitions (Krizhevsky et al., 2012). An alternative approach for calculating convolutions was proposed by Mathieu et al. (2014) who sped up the training of CNNs by calculating convolutions between batches of images and filters using fast Fourier transforms (FFTs), albeit at the cost of additional memory required for storing the filters. Recently, NVIDIA has released a library called cuDNN (Chetlur et al., 2014) that provides GPU-optimized implementations of 2D and 3D convolutions among other operations that are frequently used to implement deep learning methods, which further reduces training times compared to the previous state-of-the-art.
3.2 Overview

In this chapter, we detail our training algorithm and GPU implementation in full, with a thorough analysis of the running time on high-resolution 2D images (512 × 512) and 3D volumes (128 × 128 × 128), showing speed-ups of up to 8-fold and 200-fold, respectively for training convolutional restricted Boltzmann machines (convRBMs) and a 7-fold speed up for computing key operations for the training of CNNs compared to cuDNN. The proposed method performs training in the frequency domain, which replaces the calculation of time-consuming convolutions with simple element-wise multiplications, while adding only a small number of FFTs. In contrast to similar FFT-based approaches (e.g., Mathieu et al., 2014), our method does not use batch processing of the images as a means to reduce the number of FFT calculations, but rather minimizes FFTs even when processing a single image, which significantly reduces the required amount of scarce GPU memory. In addition, we formalize the expression of the strided convolutional deep belief network (sconvDBN), a type of convDBN that uses strided convolutions to speed up training and reduce memory requirements, in terms of stride-1 convolutions, which enables the efficient training of sconvDBNs in the frequency domain.

3.3 Algorithm

3.3.1 Training in the Spatial Domain

Before we explain the training algorithm in the frequency domain, we will briefly revise the basic steps for training convRBMs in the spatial domain. The weights and bias terms of a convRBM can be learned by contrastive divergence (CD). During each iteration of the
algorithm, the gradient of each parameter is estimated and a gradient step with a fixed learning rate is applied. The gradient of the filter weights can be approximated by

$$
\Delta w^{(ij)} \approx \frac{1}{N} (v_n^{(i)} * \tilde{h}_n^{(j)} - v_n^{(i)} * \tilde{h}_n'^{(j)}) \quad (3.1)
$$

where \(v_n, n \in [1, N]\) are images from the training set, \(h_n^{(j)}\) and \(h_n'^{(j)}\) are samples drawn from \(p(h^{(j)} | v_n)\) and \(p(h^{(j)} | v_n')\), \(v_n^{(i)} = E[v(i) | h_n]\), * denotes valid convolution, and \(\tilde{h}_n^{(j)}\) denotes a horizontal and vertical flipped version of \(h_n^{(j)}\). To apply the model to real-valued data like certain types of images, the visible units can be modelled as Gaussian units. When the visible units are mean–centred and standardized to unit variance, the expectation of the visible units is given by

$$
E[v^{(i)} | h] = \sum_{j=1}^{N_h} w^{(ij)} \odot h^{(j)} + b_i, \quad (3.2)
$$

where \(\odot\) denotes full convolution. A binary hidden unit can only encode two states. In order to increase the expressive power of the hidden units, we use noisy rectified linear units (NReLUs) as the hidden units, which have been shown to improve the learning performance of RBMs (Nair and Hinton, 2010). The hidden units can be sampled with

$$
h^{(j)} \sim \max(0, \mu^{(j)} + \mathcal{N}(0, \text{sigmoid}(\mu^{(j)}))) \quad (3.3)
$$

$$
\mu^{(j)} = \sum_{i=1}^{N_v} \tilde{w}^{(ij)} * v^{(i)} + c_j \quad (3.4)
$$

where \(\mathcal{N}(0, \sigma^2)\) denotes Gaussian noise, and \(\text{sigmoid}(x)\) is the sigmoid function defined as \(\text{sigmoid}(x) = (1 + \exp(-x))^{-1}, x \in \mathbb{R}\). The learning algorithm in the spatial domain is summarized in Figure 3.1(a).
Chapter 3 Training of Convolutional Models in the Frequency Domain

input : Images $D = \{ V_n \mid n \in [1,N] \}$
output : Weights gradient $\Delta w$

1 $\Delta w = 0$
2 foreach image $v \in D$ do
3 $v' = 0$
4 for $j = 1$ to $N_k$ do
5 $\mu^{(j)} = \sum_{i=1}^{N_c} \hat{w}^{(j)}(i) \cdot v^{(i)} + c_j$
6 $h^{(j)} \sim \max(0, \mu^{(j)} + \mathcal{N}(0, \text{sigmoid}(\mu^{(j)})))$
7 for $i = 1$ to $N_c$ do
8 $v^{(i)} = v^{(i)} + w^{(i)} \odot h^{(j)}$
9 $\forall i: v^{(i)} = v^{(i)} + b_i$
10 for $j = 1$ to $N_k$ do
11 $\mu^{(j)} = \sum_{i=1}^{N_c} \hat{w}^{(j)}(i) \cdot v^{(i)} + c_j$
12 $h^{(j)} \sim \max(0, \mu^{(j)} + \mathcal{N}(0, \text{sigmoid}(\mu^{(j)})))$
13 for $i = 1$ to $N_c$ do
14 $\Delta w^{(i)} = \Delta w^{(i)} - \hat{h}^{(j)}(i) \cdot v^{(i)}$

(a) Training in the spatial domain

input : Images $\hat{D} = \{ \hat{v}_n \mid n \in [1,N] \}$
output : Weights gradient $\Delta \hat{w}$

1 $\Delta \hat{w} = 0$
2 foreach image $\hat{v} \in \hat{D}$ do
3 $\hat{v}' = 0$
4 for $j = 1$ to $N_k$ do
5 $\hat{\mu}^{(j)} = \sum_{i=1}^{N_c} \overline{\hat{w}^{(j)}(i)} \cdot \hat{v}^{(i)}$
6 $\hat{\mu}^{(j)} = \text{ifft}(\hat{\mu}^{(j)}) + c_j$
7 $\hat{h}^{(j)} \sim \max(0, \hat{\mu}^{(j)} + \mathcal{N}(0, \text{sigmoid}(\hat{\mu}^{(j)})))$
8 $\hat{h}^{(j)} = \text{fft}(\hat{h}^{(j)})$
9 for $i = 1$ to $N_c$ do
10 $\Delta \hat{w}^{(i)} = \Delta \hat{w}^{(i)} + \overline{\hat{h}^{(j)}}(i) \cdot \hat{v}^{(i)}$
11 $\forall i: \hat{h}^{(j)}(0,0) = \hat{h}^{(j)}(0,0) + N_c^2 b_i$
12 for $j = 1$ to $N_k$ do
13 $\hat{\mu}^{(j)} = \sum_{i=1}^{N_c} \overline{\hat{w}^{(j)}(i)} \cdot \hat{v}^{(i)}$
14 $\hat{\mu}^{(j)} = \text{ifft}(\hat{\mu}^{(j)}) + c_j$
15 $\hat{h}^{(j)} \sim \max(0, \hat{\mu}^{(j)} + \mathcal{N}(0, \text{sigmoid}(\hat{\mu}^{(j)})))$
16 $\hat{h}^{(j)} = \text{fft}(\hat{h}^{(j)})$
17 for $i = 1$ to $N_c$ do
18 $\Delta \hat{w}^{(i)} = \Delta \hat{w}^{(i)} - \overline{\hat{h}^{(j)}}(i) \cdot \hat{v}^{(i)}$

(b) Training in the frequency domain

Figure 3.1: Comparison of training algorithms of convRBMs in (a) the spatial and (b) the frequency domain. Training in the frequency domain replaces the $5N_k N_c$ convolutions required in the spatial domain with simple element-wise multiplications, while adding only $4N_k$ Fourier transforms. The other operations are equivalent in both domains.

3.3.2 Training in the Frequency Domain

The computational bottleneck of the training algorithm in the spatial domain is the calculation of convolutions, which needs to be performed $5 \times N_c \times N_k$ times per iteration. To speed up the calculation of convolutions, we perform training in the frequency domain,
which maps the convolutions to simple element-wise multiplications. This is especially
important for the training on 3D images due to the relatively large number of weights of a
3D kernel compared to 2D. Calculating Fourier transforms is the most time-consuming
operation and a potential performance bottleneck, when the number of Fourier transforms
is high. To minimize the number of Fourier transforms, we map all operations needed for
training to the frequency domain whenever possible, which allows the training algorithm
to stay almost entirely in the frequency domain. This is in contrast to the FFT-based
training method developed by Mathieu et al. (2014), which only maps the calculation
of convolutions to the frequency domain and minimizes the number of FFT calculations
by processing batches of images. Unlike their method, our method performs training
primarily in the frequency domain, which reduces the number of FFT calculations even
when each image is processed sequentially, and therefore significantly reduces the memory
required for training. All of the scalar operations needed for training (multiplications and
additions) can be readily mapped to the frequency domain, because the Fourier transform
is a linear operation. Another necessary operation is the flipping of a convolutional kernel,$$
\tilde{w}(u, v) = w(N_w - u + 1, N_w - v + 1).$$ To calculate flipping efficiently, we reindex the
weights of a convolutional kernel $w^{(ij)}_{uv}$ from $u, v \in [1, N_w]$ to $u, v \in [-\lfloor N_w/2 \rfloor, (N_w - 1)/2]$, which simplifies the calculation of a flipped kernel to $\tilde{w}(u, v) = w(-u, -v)$. This
allows flipping of a kernel in the spatial domain to be expressed by the element-wise
calculation of the complex conjugate in the frequency domain, which follows directly
from the time-reversal property of the Fourier transform, i.e., if $h(x) = f(-x)$, then
$\hat{h}(\xi) = \hat{f}(\xi)$; and the reality condition, $\hat{f}(-\xi) = \hat{f}^*(\xi)$, where $\hat{x} = \mathcal{F}(x)$ denotes $x$ in the
frequency domain. Using the aforementioned mappings, equations (3.1) to (3.4) can be rewritten as

\[
\Delta \hat{w}^{(ij)} = \hat{v}^{(i)} \cdot \hat{h}^{(j)} - \hat{v}'^{(i)} \cdot \hat{h}'^{(j)} \quad (3.5)
\]

\[
E[\theta_{xy}^{(ij)} | \hat{h}] = \begin{cases} 
\sum_{j=1}^{N_k} \hat{w}^{(ij)} \hat{h}^{(j)} + N^2 b_i & \text{for } x, y = 0 \\
\sum_{j=1}^{N_k} \hat{w}^{(ij)} \hat{h}^{(j)} & \text{for } x, y \neq 0
\end{cases} \quad (3.6)
\]

\[
\hat{h}^{(j)} \sim \mathcal{F} \left( \max(0, \mathcal{F}^{-1}(\hat{\mu}^{(j)}) + c_j + \mathcal{N}(0, \sigma^2)) \right) \quad (3.7)
\]

\[
\hat{\mu}^j = \sum_{i=1}^{N_c} \hat{w}^{(ij)} \cdot \hat{v}^{(i)} \quad (3.8)
\]

where \( \sigma^2 = \text{sigm}(\mathcal{F}^{-1}(\hat{\mu}^{(j)}) + c_j) \), \( \mathcal{F}^{-1} \) denotes the inverse Fourier transform, and \( \cdot \) denotes element-wise multiplication. The algorithm for approximating the gradient in the frequency domain is summarized in Figure 3.1(b).

The only operations that cannot be directly mapped to the frequency domain are the calculation of the maximum function, the generation of Gaussian noise, and trimming of the filter kernels. To perform the first two operations, an image needs to be mapped to the spatial domain and back. However, these operations need only be calculated \( 2N_k \) times per iteration and are therefore not a significant contributor to the total running time. Because filter kernels are padded to the input image size, the size of the learned filter kernels must be explicitly enforced by trimming. This is done by transferring the filter kernels to the spatial domain, setting the values outside of the specified filter kernel size to zero, and then transforming the filter kernels back to the frequency domain. This procedure needs to be performed only once per mini-batch. Since the number of mini-batches is relatively small compared to the number of training images, trimming of the filter kernels also does
not add significantly to the total running time of the training algorithm. Although we have presented the algorithm in the context of training convRMs, the same mapping of operations can be used to speed up the training of other convolutional models, such as convolutional neural networks.

### 3.3.3 Mapping of Strided Convolutions to Stride-1 Convolutions

We have developed a method that allows convolutions with a stride \( s > 1 \) to be expressed equivalently as convolutions with stride \( s = 1 \) by reorganizing the values of \( v^{(i)} \) and \( w^{(ij)} \) to \( V^{(r)} \) and \( W^{(rf)} \) as illustrated in Figure 3.2. This reindexing scheme allows the energy function to be expressed in terms of conventional (stride-1) convolutions, which facilitates training in the frequency domain. The new indices of \( V^{(r)}_x^y \) and \( W^{(rf)}_{u,v} \) can be calculated from the old indices of \( v^{(i)}_{xy} \) and \( w^{(ij)}_{uv} \) as follows:

\[
\begin{align*}
x' &= \lfloor (x - 1)/s \rfloor + 1 \\
y' &= \lfloor (y - 1)/s \rfloor + 1 \\
i' &= s^2(i - 1) + s((y - 1) \mod s) + ((x - 1) \mod s) + 1
\end{align*}
\]  

After reorganizing \( v^{(i)} \) and \( w^{(ij)} \) to \( V^{(r)} \) and \( W^{(rf)} \), the energy of the model can be rewritten as:

\[
E(V, h) = - \sum_{i=1}^{N_c} \sum_{j=1}^{N_k} \sum_{x,y=1}^{N_h} h^{(i)}_{xy} W^{(ij)}_{u,v} V^{(i)}_{x+u-1,y+v-1} - \sum_{i=1}^{N_c} b_i \sum_{x,y=1}^{N_h} V^{(i)}_{xy} - \sum_{j=1}^{N_k} c_j \sum_{x,y=1}^{N_h} h^{(j)}_{xy}
\]

\[= - \sum_{i=1}^{N_c} \sum_{j=1}^{N_k} h^{(i)} \ast (W^{(ij)} \ast V^{(i)}) \sum_{i=1}^{N_c} b_i \sum_{x,y=1}^{N_h} V^{(i)}_{xy} - \sum_{j=1}^{N_k} c_j \sum_{x,y=1}^{N_h} h^{(j)}_{xy}
\]
**Chapter 3 Training of Convolutional Models in the Frequency Domain**

**3.3.4 GPU Implementation and Memory Considerations**

To further reduce training times, the algorithm can be efficiently implemented on graphics cards, which can perform hundreds of operations in parallel. To optimize efficiency, it is crucial to maximize the utilization of the large number of available GPU cores, while minimizing the required amount of GPU memory. Because our algorithm requires only a relatively small number of FFT calculations per iteration, the computational bottleneck is

```plaintext
where \( \cdot \) denotes the element-wise product followed by summation. The number of channels, number of visible units per channel and number of weights per channel after reorganization are given by \( N_C = N_c \times s^2 \), \( N_V^2 = N_v^2 / s^2 \) and \( N_W^2 = N_w^2 / s^2 \), respectively.

**Figure 3.2:** Illustration of convolutions with a sliding window step size \( s = 2 \) as used during filtering in a sconvDBN. A convolution of a given stride size (left side) can be efficiently calculated as the sum of multiple individual convolutions with \( s = 1 \) (right side) after rearranging the pixels of the input image and the filter kernels. The bottom row shows the actual values produced by the convolutions, which are the features from the image extracted by the filter. The figure is best viewed in colour.
the calculation of the element-wise operations, which can be performed in parallel. Thus, we distribute the processing of a single 2D image over $N_v(\lfloor N_v/2 \rfloor + 1) \times N_c$ independent threads, with one thread per element in the frequency domain. The large number of parallel threads results in a high utilization of the GPU, even when each image of a mini-batch is processed sequentially, which we do in our method, because this greatly reduces the amount of GPU memory required to store the visible and hidden units compared to processing batches of images in parallel.

Due to the relatively small amount of GPU memory available compared to CPU memory, memory requirements are an important consideration when designing a GPU algorithm. However, the total amount of required memory is highly implementation-dependent (e.g. the use of temporary variables for storing intermediate results) and a comparison can only be done with common elements at the algorithmic level. Therefore, in the remainder of this section, we focus on the memory requirements of key variables such as the visible units, the hidden units, and the filters, which are required by all implementations. In our training algorithm, all key variables are stored in the frequency domain, where each element in the frequency domain is represented by a single-precision complex number. Due to the symmetry of the Fourier space, the number of elements that need to be stored is roughly half the number of elements in the spatial domain. Thus, the memory required for storing the visible and hidden units in the frequency domain is roughly the same as in the spatial domain. A potential drawback of training in the frequency domain is that the filters need to be padded to the size of the visible units before applying the Fourier transformation, which increases the memory required for storing the filters on the GPU.
The total amount of memory in bytes required for storing the visible units, hidden units, and padded filters is given by the sum of their respective terms

$$4N_v^2N_c + \frac{4N_v^2N_k}{s^2} + 4N_v^2N_kN_c. \quad (3.14)$$

As a comparison, the training method of Krizhevsky et al. (2012) processes batches of images in order to fully utilize the GPU, which requires the storing of batches of visible and hidden units. The memory needed for storing the key variables is given by

$$4N_v^2N_bN_c + \frac{4N_v^2N_bN_k}{s^2} + 4N_v^2N_kN_c, \quad (3.15)$$

where $N_b$ is the number of images per mini-batch. Depending on the choice of the batch size, this method requires more memory for storing the visible and hidden units, while requiring less memory for storing the filters. Alternatively, Mathieu et al. (2014) proposed speeding up the training of CNNs by calculating convolutions between batches of images and filters using FFTs. The memory required for storing the visible units, hidden units, and filters using this method is given by

$$4N_v^2N_bN_c + 4N_v^2N_bN_k + 4N_v^2N_kN_c. \quad (3.16)$$

Table 3.1 shows a comparison of the memory per GPU required for storing key variables when training a network used in previous work by Krizhevsky et al. (2012). For the first layer, a comparison with Mathieu et al.’s training method could not be performed, because that method does not support strided convolutions, which would significantly reduce the memory required for storing the hidden units. In all layers, the proposed approach compensates for the increased memory requirements for the filters by considering one
Table 3.1: Comparison of the memory required for storing key variables using different training methods: our method (Freq), Krizhevsky et al.’s spatial domain method (Spat), and Mathieu et al.’s method using batched FFTs (B-FFT). A comparison with Mathieu et al.’s method could not be made for the first layer, because that method does not support strided convolutions. In all layers, our method consumes less memory for storing the key variables than the other two methods.

<table>
<thead>
<tr>
<th>Layer</th>
<th>(N_b)</th>
<th>(N_v)</th>
<th>(N_C)</th>
<th>(N_w)</th>
<th>(N_k)</th>
<th>(s)</th>
<th>Memory in MB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Freq</td>
</tr>
<tr>
<td>1</td>
<td>128</td>
<td>224</td>
<td>3</td>
<td>11</td>
<td>48</td>
<td>4</td>
<td>28.7</td>
</tr>
<tr>
<td>2</td>
<td>128</td>
<td>55</td>
<td>48</td>
<td>5</td>
<td>128</td>
<td>1</td>
<td>72.9</td>
</tr>
<tr>
<td>3</td>
<td>128</td>
<td>27</td>
<td>128</td>
<td>3</td>
<td>192</td>
<td>1</td>
<td>69.2</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>13</td>
<td>192</td>
<td>3</td>
<td>192</td>
<td>1</td>
<td>24.0</td>
</tr>
<tr>
<td>5</td>
<td>128</td>
<td>13</td>
<td>192</td>
<td>3</td>
<td>128</td>
<td>1</td>
<td>16.1</td>
</tr>
</tbody>
</table>

image at a time rather than a batch, and still outperforms batched learning in the spatial domain in terms of speed (see Section 3.4), despite of not using batches.

### 3.4 Evaluation of Running Time

#### 3.4.1 Comparison of Running Times for Training sconvDBNs

To demonstrate where the performance gains are produced, we trained a two-layer sconvDBN on 2D and 3D images using our frequency-domain method and the following methods that all compute convolutions on the GPU, but using different approaches: 1) our spatial domain implementation that convolves a single 2D or 3D image with a single 2D or 3D filter kernel at a time, 2) Krizhevsky’s spatial domain convolution implementation (Krizhevsky, 2012), which is a widely used method (e.g., Hinton and Srivastava, 2012; Scherer et al., 2010; Zeiler and Fergus, 2013) that calculates the convolution of batches of 2D images and 2D filter kernels in parallel (note that this method cannot be applied to 3D images, so it was only used for the 2D experiments), and 3) our implementation that
calculates convolutions using FFTs, but without mapping the other operations that would allow the algorithm to stay in the frequency domain when not computing convolutions. The parameters that we used for training convRBMs on 2D and 3D images are summarized in Table 3.2. The key parameters that we varied for our experiments are the filter size and stride size of the first layer, and the filter size and the number of channels of the second layer. Because the number of channels of the second layer is equal to the number of filters of the first layer, we also varied the number of filters of the first layer in order to attain the desired number of channels. For all implementations, the training time is directly proportional to the number of filters. Therefore, a detailed comparison of all four methods with a varying number of filters was not performed. The hardware details of our test environment are summarized in Table 3.3.

**Table 3.2:** Training parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>ImageNet (2D)</th>
<th>OASIS (3D)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1st layer</td>
<td>2nd layer</td>
</tr>
<tr>
<td>Filter size</td>
<td>5 to 52</td>
<td>5 to 13</td>
</tr>
<tr>
<td>Stride size</td>
<td>1, 2, 4</td>
<td>1</td>
</tr>
<tr>
<td>Number of channels</td>
<td>3</td>
<td>16, 32, 64</td>
</tr>
<tr>
<td>Number of filters</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Image dimension</td>
<td>512&lt;sup&gt;2&lt;/sup&gt;</td>
<td>128&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
<tr>
<td>Number of images</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>Batch size</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

**Table 3.3:** Hardware specification of our test system for training sconvDBNs using different implementations for calculating convolutions.

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel i7-3770 CPU @ 3.40 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Memory</td>
<td>8 GB</td>
</tr>
<tr>
<td>Graphics Card</td>
<td>NVIDIA GeForce GTX 660</td>
</tr>
<tr>
<td>GPU Cores</td>
<td>960 cores @ 1.03 GHz</td>
</tr>
<tr>
<td>GPU Memory</td>
<td>2 GB</td>
</tr>
</tbody>
</table>
For the comparison on 2D images, we used a data set of 256 natural colour images from the ImageNet data set (DENG et al., 2009). All images were resampled to a resolution of $512 \times 512$ pixels per colour channel. For the evaluation on 3D images, we used 100 magnetic resonance imaging (MRI) scans of the brain from the Open Access Series of Imaging Studies (OASIS) data set (MARCUS et al., 2007). We resampled all volumes to a resolution of $128 \times 128 \times 128$ voxels and a voxel size of $2 \times 2 \times 2$ mm.

**Running Time Analysis on 2D Colour Images (ImageNet)**

Figure 3.3(a) shows a comparison of running times for training the first sconvRBM layer on 256 images with varying filter and stride sizes. Due to internal limitations of Krizhevsky’s convolution implementation, it cannot be applied to images with a resolution of $512 \times 512$ pixels when using a stride size smaller than four, and those comparisons could not be made. Our frequency domain implementation is between 2 to 24 times faster than our convolution implementation, where the speed gains are larger for larger filter and stride sizes. The running time of the methods that calculate convolutions in the frequency domain ($\text{fft}$ and $\text{freq}$) are independent of the filter size, because the filters need to be padded to the size of the input images before calculating the FFT and therefore the same number of operations are required for training for all filter sizes. For a stride of one, the impact of the convolution implementation on the total running time is relatively low, because the computational bottleneck is the inference and sampling of the hidden units. As the number of hidden units decreases with larger strides, the running time becomes more dependent on the time spent to calculate convolutions. Hence, the differences between the four methods are more pronounced for larger strides. For a stride of four, training in the frequency domain is between 8 to 24 times faster than training in the spatial domain using our convolution implementation and 2 to 7 times faster than using batched
(a) Running times of training a first layer sconvRBM with stride sizes of 1, 2, and 4.

(b) Running times of training a second layer sconvRBM with 16, 32, and 64 channels (stride size 1).

**Figure 3.3:** Comparison of running times for training a (a) first and (b) second layer sconvRBM on 2D images using our frequency domain method (freq) and three alternative methods using different convolution implementations: single image convolutions (spat1), batched convolutions (spat2), and convolution by using FFTs (fft). Due to internal limitations of the implementation of batched convolutions, a comparison with spat2 could not be performed for images with a resolution of 512 × 512 when using a stride size smaller than four.

Calculating convolutions by FFTs is the slowest method for all stride sizes and 2D filter sizes up to 44, largely due to the cost of calculating Fourier transforms.

Figure 3.3(b) shows a similar comparison for training the second convRBM layer for a stride size of one and varying filter sizes and numbers of channels. In contrast to training the first layer, training times mostly depend on the calculation of convolutions, where the
impact of calculating convolutions on the total running time increases with an increasing number of channels. Training in the frequency domain is between 5 to 26 times faster than training in the spatial domain using single-image convolutions, and 2 to 8 times faster than using batched convolutions. For all channel sizes, batched training is about 3 to 4 times faster than non-batched training and calculating convolutions using FFTs is much slower than batched training and training in the frequency domain. To summarize, training of 2D images in the frequency domain is much faster than training in the spatial domain even for small filter sizes. Using the largest filter kernels in both layers, the proposed method is shown to yield a speedup of 7 to 8 times compared to state-of-the-art GPU implementations.

**Running Time Analysis on 3D Volumes (OASIS)**

Figure 3.4 shows the comparison of running times for training a first and second layer sconvRBM on 3D volumes for varying filter sizes, stride sizes, and varying numbers of channels. In contrast to training on 2D images, the computational costs of calculating 3D convolutions break even with calculating FFTs even for small filter sizes, because the number of multiplications and additions per convolution increases cubically, instead of quadratically, with the filter kernel size. As a result, simply training by convolutions in the frequency domain is faster than in the spatial domain. However, our proposed training algorithm still outperforms both other methods, even at the smallest filter size. For filter sizes of five and larger, our frequency domain implementation is between 3.5 to 200 times faster than our spatial domain implementation using single-image convolutions and 2.7 to 17 times faster than calculating convolutions by FFTs. Similar to the results on 2D images, training times of the first layer using a stride of one depend strongly on the time required to calculate the expectation of the hidden units and to sample the hidden units. Hence,
Figure 3.4: Comparison of running times for training a (a) first and (b) second layer sconvRBM on 3D volumes using a single 3D image convolution implementation (spat), an implementation that calculates convolutions by using FFTs (fft), and our proposed implementation in the frequency domain (freq).

Performance improvements of our frequency domain method are more pronounced for larger strides and numbers of channels, where the impact of calculating convolutions on the total training time is also larger. This makes the proposed method particularly suitable for training sconvRBMs on high-resolution 3D volumes.
3.4.2 Comparison of Running Times for Calculating Convolutions with cuDNN

The NVIDIA CUDA Deep Neural Network library (cuDNN) (Chetlur et al., 2014) is a GPU-accelerated library of primitives that are commonly required to implement deep learning methods such as CNNs and convDBNs. It is used internally by state-of-the-art deep learning frameworks such as Caffe (Jia et al., 2014), Theano (Bastien et al., 2012), and Torch (Collobert et al., 2011a). The library in its 7th version provides highly optimized implementations for calculating, e.g., batched 2D and 3D convolutions, pooling operations, and different transfer functions. In this set of experiments, we directly compared the time required to calculate convolutions when training a CNN using our frequency domain implementation with cuDNN. We only consider convolutions because they are the most time-consuming operations and because all other operations are calculated in the same way. For the comparison with our frequency domain implementation, the time measurements also include the time required to calculate FFTs during the gradient calculation and for transforming the accumulated gradient to spatial domain and back in order to trim the padded filters to the original filter size. The parameters that we used for evaluating the running time of different convolution implementations are summarized in Table 3.4. The parameters represent typical values of the first and second layer of a CNN, which are the layers that require the most time to train, because the visible units of subsequent layers are commonly of much smaller resolution. The key parameters that we varied are the image size, filter size, and batch size. The hardware details of our test environment are summarized in Table 3.5.

Figure 3.5 shows a comparison of the running times for calculating convolutions using our frequency domain implementation and cuDNN for varying image sizes. The FFT implementation that we used internally, cuFFT, is optimized for image sizes that can be
factorized as $2^a \times 3^b \times 5^c$. Therefore, we also evaluated the time required to calculate convolutions when the input images are automatically padded to a size for which cuFFT has been optimized. For image sizes larger than $110^3$ voxels, cuFFT requires significantly more temporary memory for calculating FFTs of unoptimized image sizes. Consequently, only the running times of the padded frequency domain implementation and cuDNN were measured for image sizes larger than $110^3$ voxels. Calculating convolutions in the frequency domain scales better with an increase of the image size than cuDNN. While cuDNN is the fastest method for very small image sizes, the padded frequency domain implementation is more than 20 times faster for images with a size of $128^3$ voxels and 2 input channels, and more than 18 times faster for images with a size of $64^3$ voxels and 32 channels. Calculating the FFT and therefore the running time of the frequency domain implementation varies significantly depending on the input image size, and careful padding of the input images is required to achieve consistently high performance.
Chapter 3 Training of Convolutional Models in the Frequency Domain

Figure 3.5: Comparison of running times of key operations for training a single CNN layer for varying number of channels and input sizes. Frequency domain training and training using cuDNN scales comparable for small numbers of channels. For large numbers of channels, frequency domain method scales slightly better to larger images.

Table 3.6 shows a comparison of running times per image for varying batch sizes and for varying number of channels. Increasing the batch size has only a minor effect on the running time of cuDNN, while significantly increasing the required amount of GPU memory. In contrast, the implementation in the frequency domain does not require more memory, because each image of a mini-batch is processed sequentially. Increasing the batch size reduces the average running time per image of our frequency domain implementation, because the FFTs required to trim the padded filters to the original filter size need to be calculated only once per mini-batch, and have therefore a smaller impact on the overall running time when the number of images per mini-batch is larger.

The impact of the filter size on the running time is shown in Figure 3.6. For the first layer, training in the frequency domain is faster for all filter sizes, where the speed-ups are larger for larger filters. At a filter size of 9, the speed-up is 20 times. For 32 channels, cuDNN is faster for very small filter sizes and training in the frequency domain breaks
Table 3.6: Comparison of running times for calculating key operations for training a CNN layer for different batch sizes. Increasing the batch size reduces the impact of cropping the learned filters on the overall running time and consequently reduces the average time to process one image. The cuDNN implementation only benefits mildly from using larger batches.

<table>
<thead>
<tr>
<th>Batch size</th>
<th>2 channels</th>
<th>32 channels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Freq</td>
<td>cuDNN</td>
</tr>
<tr>
<td>1</td>
<td>0.109</td>
<td>1.397</td>
</tr>
<tr>
<td>2</td>
<td>0.081</td>
<td>1.011</td>
</tr>
<tr>
<td>4</td>
<td>0.068</td>
<td>1.249</td>
</tr>
<tr>
<td>8</td>
<td>0.060</td>
<td>1.247</td>
</tr>
</tbody>
</table>

Figure 3.6: Comparison of running times of key operations for training a single CNN layer for varying number of channels and filter sizes.

even at a filter size of 3. For filter sizes larger than 3, training in the frequency domain is substantially faster with a speed-up of up to 18 times for a filter size of 9.

Table 3.7 shows a comparison of running times for calculating convolutions of our frequency domain implementation with cuDNN for the 7-layer convolutional encoder network (CEN) that we use in Chapter 4 to segment multiple sclerosis (MS) lesions, and the first sconvRBM of the DBN model described in Chapter 5 to model the variability in lesion distribution and brain morphology. For the 7-layer CEN, the second convolutional and the first deconvolutional layer benefits the most from training in the frequency domain
Table 3.7: Comparison of running times of time critical operations of the 7-layer CEN-s used for segmenting lesions, and the first sconvRBM of the lesion DBN using to model lesion distribution. The running times of pooling layers we excluded.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Running time [s]</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>7-layer CEN used for lesion segmentation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1st convolutional layer</td>
<td>0.077</td>
<td>0.498</td>
</tr>
<tr>
<td>2nd convolutional layer</td>
<td>0.052</td>
<td>0.524</td>
</tr>
<tr>
<td>1st deconvolutional layer</td>
<td>0.052</td>
<td>0.524</td>
</tr>
<tr>
<td>2nd deconvolutional layer</td>
<td>0.148</td>
<td>0.517</td>
</tr>
<tr>
<td>total</td>
<td>0.328</td>
<td>2.063</td>
</tr>
<tr>
<td><strong>First sconvRBM of the lesion DBN</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>direct calculation of strided convolutions</td>
<td>0.076</td>
<td>0.068</td>
</tr>
<tr>
<td>strided convolutions mapped to stride-1 convolutions</td>
<td>0.019</td>
<td>0.071</td>
</tr>
</tbody>
</table>

due to the relatively large number of channels in these layers. Overall, the convolutions required to train the 7-layer CEN can be calculated 6.3 times faster in the frequency domain compared to cuDNN. In a second experiment, we compared the time required to directly calculate strided convolutions with mapping strided convolutions to stride-1 convolutions by reorganizing the visible units of an sconvRBM. For the frequency domain implementation, the direct method calculates stride-1 convolutions instead of strided convolutions and discards the additional hidden units afterwards, because strided convolutions cannot directly be calculated in the frequency domain. Mapping strided convolutions to stride-1 convolutions speeds up the training in the frequency domain by a factor of four compared to calculating stride-1 convolutions in the frequency domain. Overall, calculating the convolutions for training the first sconvRBM is 3.7 times faster using our frequency domain implementation than the direct implementation of strided convolutions of cuDNN.
3.5 Conclusions

We have presented a fast training method for convolutional models, which performs training in the frequency domain in order to replace the time-consuming computation of convolutions with simple element-wise multiplications. We have shown that it is also essential to map the other operations to frequency space wherever possible to minimize the number of Fourier transforms, and that this greatly decreases training times over performing only the convolutions in the frequency domain. In addition, our method can be efficiently implemented on the GPU and is faster than a highly optimized GPU implementation of batched convolutions in the spatial domain. We have evaluated the running time improvements using two standard benchmark data sets, showing a speed-up of up to 8 times on 2D images from the ImageNet data set and up to 200 times on 3D volumes from the OASIS data set. In addition, we have directly compared the time required to calculate convolutions using our method with cuDNN, the current state-of-the-art library for calculating 2D and 3D convolutions, with the results showing that our method can calculate convolutions up to 20 times faster than cuDNN.
Chapter 4

White Matter Lesion Segmentation

4.1 Introduction

Multiple sclerosis (MS) is an inflammatory and demyelinating disease of the central nervous system with pathology that can be observed in vivo by magnetic resonance imaging (MRI). MS is characterized by the formation of lesions, primarily visible in the white matter on conventional MRI. Imaging biomarkers based on the delineation of lesions, such as lesion load and lesion count, have established their importance for assessing disease progression and treatment effect. However, lesions vary greatly in size, shape, intensity and location, which makes their automatic and accurate segmentation challenging.

4.2 Related Work

Many automatic methods have been proposed for the segmentation of MS lesions over the last two decades (GARCÍA-LORENZO et al., 2013), which can be classified into unsupervised and supervised methods.
4.2.1 Unsupervised Methods

Unsupervised methods do not require a labeled data set for training. Instead, lesions are identified as an outlier of, e.g., a subject-specific generative model of tissue intensities (Roura et al., 2015; Schmidt et al., 2012; Tomas-Fernandez and Warfield, 2015; Van Leemput et al., 2001), or a generative model of image patches representing a healthy population (Weiss et al., 2013). Alternatively, clustering methods have been used to segment healthy and lesion tissue, where lesions are modelled as a separate tissue class (Shiee et al., 2010; Sudre et al., 2015). In many methods, spatial priors of healthy tissues are used to reduce false positives. For example, in addition to modelling MS lesions as a separate intensity cluster, Lesion-TOADS (Shiee et al., 2010) employs topological and statistical atlases to produce a topology-preserving segmentation of all brain tissues.

To account for local changes of the tissue intensity distributions, Tomas-Fernandez and Warfield (2015) combined the subject-specific model of the global intensity distributions with a voxel-specific model calculated from a healthy population, where lesions are detected as outliers of the combined model. A major challenge of unsupervised methods is that outliers are often not specific to lesions and can also be caused by intensity inhomogeneities, partial volume, imaging artifacts, and small anatomical structures such as blood vessels, which leads to the generation of false positives. To overcome these limitations, Roura et al. (2015) employed an additional set of rules to remove false positives, while Schmidt et al. (2012) used a conservative threshold for the initial detection of lesions, which are later grown in a separate step to yield an accurate delineation.
4.2.2 Supervised Methods

Current supervised approaches typically start with a set of features, which can range from small and simple to large and highly variable, and are either predefined by the user (Geremia et al., 2011, 2010; Guizard et al., 2015; Subbanna et al., 2015) or gathered in a feature extraction step such as by deep learning (Yoo et al., 2014). Voxel-based segmentation algorithms (Geremia et al., 2010; Yoo et al., 2014) feed the features and labels of each voxel into a general classification algorithm, such as a random forest (Breiman, 2001), to classify each voxel and to determine which set of features are the most important for segmentation in the particular domain. Voxel features and the labels of neighboring voxels can be incorporated into Markov random field-based (MRF-based) approaches (Subbanna et al., 2015, 2009) to produce a spatially consistent segmentation. As a strategy to reduce false positives, Subbanna et al. (2015) combined a voxel-level MRF with a regional MRF, which integrates a large set of intensity and textural features extracted from the regions produced by the voxel-level MRF with the labels of neighboring nodes of the regional MRF. Library-based approaches leverage a library of pre-segmented images to carry out the segmentation. For example, Guizard et al. (2015) proposed a segmentation method based on an extension of the non-local means algorithms (Coupe et al., 2011). The centres of patches at every voxel location are classified based on matched patches from a library containing pre-segmented images, where multiple matches are weighted using a similarity measure based on rotation-invariant features.

4.2.3 Patch-based Deep Learning Methods

A recent breakthrough for the automatic image segmentation using deep learning comes from the domain of cell membrane segmentation, in which Ciresan et al. (2012) proposed
Chapter 4  White Matter Lesion Segmentation

classifying the centres of image patches directly using a convolutional neural network (CNN) (LeCun et al., 1998) without a dedicated feature extraction step. Instead, features are learned indirectly within the lower layers of the neural network during training, while the higher layers can be regarded as performing the classification, which allows the learning of features that are specifically tuned to the segmentation task. However, the time required to train patch-based methods can make the approach infeasible when the size and number of patches are large.

4.2.4 Fully Convolutional Methods

Recently, different CNN architectures (Brosch et al., 2015; Kang and Wang, 2014; Li et al., 2014; Long et al., 2015; Ronneberger et al., 2015) have been proposed that are able to feed through entire images, which removes the need to select representative patches, eliminates redundant calculations where patches overlap, and therefore these models scale up more efficiently with image resolution. Kang and Wang (2014) introduced the fully convolutional neural network (fCNN) for the segmentation of crowds in surveillance videos. However, fCNNs produce segmentations of lower resolution than the input images due to the successive use of convolutional and pooling layers, both of which reduce the dimensionality. To predict segmentations of the same resolution as the input images, we recently proposed using a 3-layer convolutional encoder network (CEN) (Brosch et al., 2015) for MS lesion segmentation. The combination of convolutional (LeCun et al., 1998) and deconvolutional (Zeiler et al., 2011) layers allows our network to produce segmentations that are of the same resolution as the input images.

Another limitation of the traditional CNN is the trade-off between localization accuracy, represented by lower-level features, and contextual information, provided by higher-level features. To overcome this limitation, Long et al. (2015) proposed fusing the segmentations
produced by the lower layers of the network with the upsampled segmentations produced by higher layers. However, using only low-level features was not sufficient to produce a good segmentation at the lowest layers, which is why segmentation fusion was only performed for the three highest layers. Instead of combining the segmentations produced at different layers, RONNEBERGER et al. (2015) proposed combining the features of different layers to calculate the final segmentation directly at the lowest layer using an 11-layer u-shaped network architecture called u-net. Their network is composed of a traditional contracting path (first half of the u), but augmented with an expanding path (last half of the u), which replaces the pooling layers of the contracting path with upsampling operations. To leverage both high- and low-level features, shortcut connections are added between corresponding layers of the two paths. However, upsampling cannot fully compensate for the loss of resolution, and special handling of the border regions is still required.

4.3 Methods

We propose a new convolutional network architecture that combines the advantages of a CEN (BROSCH et al., 2015) and a u-net (RONNEBERGER et al., 2015). Our network is divided into two pathways, a traditional convolutional pathway, which consists of alternating convolutional and pooling layers, and a deconvolutional pathway, which consists of alternating deconvolutional and unpooling layers (ZEILER et al., 2011) and predicts the final segmentation. Similar to the u-net, we introduce shortcut connections between layers of the two pathways. In contrast to the u-net, our method produces segmentations of the same resolution as the input images, independent of the number of layers and filter sizes used, which enables the use of deeper networks without requiring special handling of the border regions.
Figure 4.1: Pre-training and fine-tuning a 7-layer convolutional encoder network with shortcut. The activations calculated by deconvolution through a shortcut are added to the corresponding layer activations before applying the transfer function. For deeper network architectures, there is a shortcut between all convolutional and deconvolutional layers except for the top-most layer. Pre-training is performed on the input images using a stack of convolutional RBMs. The pre-trained weights and bias terms are used to initialize a CEN, which is fine-tuned on pairs of input images, $x(0)$, and segmentations, $y(0)$.

The task of segmenting MS lesions is defined as finding a function $s$ that maps multi-modal images $I$, e.g., $I = (I_{FLAIR}, I_{T1})$, to corresponding binary lesion masks $S$, where 1 denotes a lesion voxel and 0 denotes a non-lesion voxel. Given a set of training images $I_n$, $n \in \mathbb{N}$, and corresponding segmentations $S_n$, we model finding an appropriate function for segmenting MS lesions as an optimization problem of the following form

$$s = \arg \min_{s \in S} \sum_n E(S_n, s(I_n)),$$  \hspace{1cm} (4.1)

where $S$ is the set of possible segmentation functions, and $E$ is an error measure that calculates the dissimilarity between ground truth segmentations and predicted segmenta-
tions. The set of possible segmentation functions \( S \) is defined by the network architecture, where a particular function \( s \) is defined by the combination of network architecture and learned parameters of the network.

### 4.3.1 Model Architecture

The set of possible segmentation functions, \( S \), is modelled by the convolutional encoder network with shortcut connections (CEN-s) illustrated in Figure 4.1. A CEN-s is a type of CNN (LeCun et al., 1998) that is divided into two interconnected pathways, the convolutional pathway and the deconvolutional (Zeiler et al., 2011) pathway. The convolutional pathway consists of alternating convolutional and pooling layers. The input layer of the convolutional pathway is composed of the image voxels \( x_i^{(0)}(p) \), \( i \in [1, C] \), where \( i \) indexes the modality or input channel, \( C \) is the number of modalities or channels, and \( p \in \mathbb{N}^3 \) are the coordinates of a particular voxel. The convolutional layers automatically learn a feature hierarchy from the input images. A convolutional layer is a deterministic function of the following form

\[
x_j^{(l)} = \max \left( 0, \sum_{i=1}^{C} \tilde{w}_{c,ij}^{(l)} * x_i^{(l-1)} + b_j^{(l)} \right),
\]

where \( l \) is the index of a convolutional layer, \( x_j^{(l)} \), \( j \in [1, F] \), denotes the feature map corresponding to the trainable convolution filter \( \tilde{w}_{c,ij}^{(l)} \), \( F \) is the number of filters of the current layer, \( b_j^{(l)} \) are trainable bias terms, \( * \) denotes valid convolution, and \( \tilde{w} \) denotes a flipped version of \( w \), i.e., \( \tilde{w}(a) = w(-a) \). To be consistent with the inference rules of convolutional restricted Boltzmann machines (convRBM) (Lee et al., 2009), which are used for pre-training, convolutional layers convolve the input signal with flipped filter kernels, while deconvolutional layers calculate convolutions with non-flipped filter kernels.
We use rectified linear units (NAIR and HINTON, 2010) in all layers except for the output layers, which have shown to improve the classification performance of CNNs (KRIZHEVSKY et al., 2012). A convolutional layer is followed by an average pooling layer (SCHERER et al., 2010) that halves the number of units in each dimension by calculating the average of each block of $2 \times 2 \times 2$ units per channel.

The deconvolutional pathway consists of alternating deconvolutional and unpooling layers with shortcut connections to the corresponding convolutional layers. The first deconvolutional layer uses the extracted features of the convolutional pathway to calculate abstract segmentation features

$$y^{(L-1)}_i = \max \left( 0, \sum_{j=1}^{F} w^{(L)}_{d,ij} \ast y^{(L)}_j + c^{(L-1)}_i \right), \quad (4.3)$$

where $y^{(L)} = x^{(L)}$, $L$ denotes the number of layers of the convolutional pathway, $w^{(L)}_{d,ij}$ and $c^{(L-1)}_i$ are trainable parameters of the deconvolutional layer, and $\ast$ denotes full convolution. To be consistent with the general notation of deconvolutions (ZEILER et al., 2011), the non-flipped version of $w$ is convolved with the input signal.

Subsequent deconvolutional layers use the activations of the previous layer and corresponding convolutional layer to calculate more localized segmentation features

$$y^{(l)}_i = \max \left( 0, \sum_{j=1}^{F} w^{(l+1)}_{d,ij} \ast y^{(l+1)}_j + \sum_{j=1}^{F} w^{(l+1)}_{s,ij} \ast x^{(l+1)}_j + c^{(l)}_i \right), \quad (4.4)$$

where $l$ is the index of a deconvolutional layer with shortcut, and $w^{(l+1)}_{s,ij}$ are the shortcut filter kernels connecting the activations of the convolutional pathway with the activations of the deconvolutional pathway. The last deconvolutional layer integrates the low-level
features extracted by the first convolutional layer with the high-level features from the previous layer to calculate a probabilistic lesion mask

\[ y^{(0)}_1 = \text{sigm} \left( \sum_{j=1}^{F} \left( w_{d,1j}^{(1)} \otimes y^{(1)}_j + w_{s,1j}^{(1)} \otimes x^{(1)}_j \right) + c_1^{(0)} \right), \]  

(4.5)

where we use the sigmoid function defined as \( \text{sigm}(z) = \left(1 + \exp(-z)\right)^{-1}, z \in \mathbb{R} \), instead of the rectified linear function in order to obtain a probabilistic segmentation with values in the range between 0 and 1. To produce a binary lesion mask from the probabilistic output of our model, we chose a fixed threshold such that the mean Dice similarity coefficient (DSC) (Dice, 1945) is maximized on the training set and used the same threshold for the evaluation on the test set.

### 4.3.2 Gradient Calculation

The parameters of the model can be efficiently learned by minimizing the error \( E \) for each sample of the training set, which requires the calculation of the gradient of \( E \) with respect to the model parameters (LeCun et al., 1998). Typically, neural networks are trained by minimizing either the sum of squared differences (SSD) or the cross-entropy. Here, we will derive the gradient on the example of the SSD, due to the similarity of the SSD with the later proposed objective function. The SSD can be calculated for a single image as follows

\[ E = \frac{1}{2} \sum_{p} \left( S(p) - y^{(0)}(p) \right)^2, \]  

(4.6)
where \( p \in \mathbb{N}^3 \) are the coordinates of a particular voxel. The partial derivatives of the error with respect to the model parameters can be calculated using the delta rule and are given by

\[
\frac{\partial E}{\partial w_{d,ij}^{(l)}} = \delta_{d,j}^{(l-1)} \ast y_j^{(l)}, \quad \frac{\partial E}{\partial c_i^{(l)}} = \sum_p \delta_{d,ij}^{(l)}(p), \tag{4.7}
\]

\[
\frac{\partial E}{\partial w_{s,ij}^{(l)}} = \delta_{d,i}^{(l-1)} \ast \tilde{x}_j^{(l)}, \tag{4.8}
\]

\[
\frac{\partial E}{\partial w_{c,ij}^{(l)}} = x_i^{(l-1)} \ast \delta_{c,j}^{(l)}, \quad \text{and} \quad \frac{\partial E}{\partial b_i^{(l)}} = \sum_p \delta_{c,ij}^{(l)}(p). \tag{4.9}
\]

For the output layer, \( \delta_{d,1}^{(0)} \) can be calculated by

\[
\delta_{d,1}^{(0)} = (y_1^{(0)} - S)y_1^{(0)}(1 - y_1^{(0)}). \tag{4.10}
\]

The derivatives of the error with respect to the parameters of the other layers can be calculated by applying the chain rule of partial derivatives, which yields to

\[
\delta_{d,j}^{(l)} = \sum_i C \left( \tilde{w}_{d,ij}^{(l)} \ast \delta_{d,i}^{(l-1)} \right) \mathbb{I}(y_j^{(l)} > 0), \tag{4.11}
\]

\[
\delta_{c,j}^{(l)} = \sum_j F \left( w_{c,ij}^{(l+1)} \ast \delta_{c,j}^{(l+1)} \right) \mathbb{I}(x_i^{(l)} > 0), \tag{4.12}
\]

where \( l \) is the index of a deconvolutional or convolutional layer, \( \delta_{c,i}^{(L)} = \delta_{d,i}^{(L)} \), and \( \mathbb{I}(z) \) denotes the indicator function defined as 1 if the predicate \( z \) is true and 0 otherwise. If a
layer is connected through a shortcut, \( \delta_{c,j}^{(l)} \) needs to be adjusted by propagating the error back through the shortcut connection. In this case, \( \delta_{c,j}^{(l)} \) is calculated by

\[
\delta_{c,j}^{(l)} = (\delta_{c,j}^{(l)} + \hat{w}_{s,j}^{(l)} \ast \delta_{d,j}^{(l-1)}) \mathbb{I}(x_j^{(l)} > 0),
\]

(4.13)

where \( \delta_{c,j}^{(l)} \) denotes the activation of unit \( \delta_{c,j}^{(l)} \) before taking the shortcut connection into account.

The sum of squared differences is a good measure of classification accuracy, if the two classes are fairly balanced. However, if one class contains vastly more samples, as is the case for lesion segmentation, the error measure is dominated by the majority class and consequently, the neural network would learn to ignore the minority class. To overcome this problem, we use a combination of sensitivity and specificity, which can be used together to measure classification performance even for vastly unbalanced problems. More precisely, the final error measure is a weighted sum of the mean squared difference of the lesion voxels (sensitivity) and non-lesion voxels (specificity), reformulated to be error terms:

\[
E = r \frac{\sum_p (S(p) - y^{(0)}(p))^2 \cdot S(p)}{\sum_p S(p)} + (1 - r) \frac{\sum_p (S(p) - y^{(0)}(p))^2 \cdot (1 - S(p))}{\sum_p (1 - S(p))}.
\]

(4.14)

We formulate the sensitivity and specificity errors as squared errors in order to yield smooth gradients, which makes the optimization more robust. The sensitivity ratio \( r \) can be used to assign different weights to the two terms. Due to the large number of non-lesion voxels, weighting the specificity error higher is important, but based on preliminary experimental results (Brosch et al., 2015), the algorithm is stable with respect to changes in \( r \), which largely affects the threshold used to binarize the probabilistic output. A
detailed evaluation of the impact of the sensitivity ratio on the learned model is presented in Section 4.4.4.

To train our model, we must compute the derivatives of the modified objective function with respect to the model parameters. Equations (4.7)–(4.9) and (4.11)–(4.13) are a consequence of the chain rule and independent of the chosen similarity measure. Hence, we only need to derive the update rule for \( \delta_{d,1}^{(0)} \). With \( \alpha = 2r(\sum_p S(p))^{-1} \) and \( \beta = 2(1 - r)(\sum_p (1 - S(p)))^{-1} \), we can rewrite \( E \) as

\[
E = \frac{1}{2} \sum_p \left( S(p) - y_1^{(0)}(p) \right)^2 \alpha S(p) + \frac{1}{2} \sum_p \left( S(p) - y_1^{(0)}(p) \right)^2 \beta (1 - S(p)) \tag{4.15}
\]

\[
= \frac{1}{2} \sum_p \left( \alpha S(p) + \beta (1 - S(p)) \right) \left( S(p) - y_1^{(0)}(p) \right)^2. \tag{4.16}
\]

Our objective function is similar to the SSD, with an additional multiplicative term applied to the squared differences. The additional factor only depends on the target segmentation \( S \) and is therefore constant with respect to the model parameters. Consequently, \( \delta_{d,1}^{(0)} \) can be derived analogously to the SSD case, and the new factor is simply carried over:

\[
\delta_{d,1}^{(0)} = (\alpha S + \beta (1 - S))(y_1^{(0)} - S)y_1^{(0)} (1 - y_1^{(0)}). \tag{4.17}
\]

4.3.3 Training

At the beginning of the training procedure, the model parameters need to be initialized and the choice of the initial parameters can have a big impact on the learned model (Sutskever et al., 2013). In our experiments, we found that initializing the model using pre-training (Hinton and Salakhutdinov, 2006) on the input images was required in order to be able to fine-tune the model using the ground truth segmentations without getting stuck
early in a local minimum. Pre-training can be performed layer by layer (Hinton et al., 2006) using a stack of convRBMs (see Figure 4.1), thereby avoiding the potential problem of vanishing or exploding gradients (Hochreiter, 1991). The first convRBM is trained on the input images, while subsequent convRBMs are trained on the hidden activations of the previous convRBM. After all convRBMs have been trained, the model parameters of the CEN-s can be initialized as follows (showing the first convolutional and the last deconvolutional layers only, see Figure 4.1)

\[
\begin{align*}
    w_c^{(1)} &= \hat{w}^{(1)}, & w_d^{(1)} &= 0.5\hat{w}^{(1)}, & w_s^{(1)} &= 0.5\hat{w}^{(1)} \\
    b^{(1)} &= \hat{b}^{(1)}, & c^{(0)} &= c^{(1)},
\end{align*}
\] (4.18) (4.19)

where \(\hat{w}^{(1)}\) are the filter weights, \(\hat{b}^{(1)}\) are the hidden bias terms, and \(c^{(1)}\) are the visible bias terms of the first convRBM.

A major challenge for gradient-based optimization methods is the choice of an appropriate learning rate. Classic stochastic gradient descent (LeCun et al., 1998) uses a fixed or decaying learning rate, which is the same for all parameters of the model. However, the partial derivatives of parameters of different layers can vary substantially in magnitude, which can require different learning rates. In recent years, there has been an increasing interest in developing methods for automatically choosing independent learning rates. Most methods (e.g., AdaGrad by Duchi et al., 2011; AdaDelta by Zeiler, 2012; RMSprop by Dauphin et al., 2015; and Adam by Kingma and Ba, 2014) collect different statistics of the partial derivatives over multiple iterations and use this information to set an adaptive learning rate for each parameter. This is especially important for the training of deep networks, where the optimal learning rates often differ greatly for each layer. In
our initial experiments, networks obtained by training with AdaDelta, RMSprop, and Adam performed comparably well, but AdaDelta was the most robust to the choice of hyperparameters, so we used AdaDelta for all results reported.

### 4.3.4 Implementation

Pre-training and fine-tuning were performed using the highly optimized GPU-accelerated implementation of 3D convRBMs and CNNs (Brosch and Tam, 2015) explained in Chapter 3. Our frequency domain implementation significantly speeds up the training by mapping the calculation of convolutions to simple element-wise multiplications, while adding only a small number of Fourier transforms. This is especially beneficial for the training on 3D volumes, due to the increased number of weights of 3D kernels compared to 2D. Although GPU-accelerated deep learning libraries based on the NVIDIA CUDA Deep Neural Network library (cuDNN) (Chetlur et al., 2014) are publicly available (e.g., Bastien et al., 2012; Collobert et al., 2011a; Jia et al., 2014), we trained our models using our own implementation, because we optimized it for memory facilitating the training on large 3D volumes and because it performs the most computationally intensive training operations six times faster than cuDNN in a direct comparison (see Table 3.7 on page 49).

### 4.4 Experiments and Results

We evaluated our method on two publicly available data sets (from the MICCAI 2008 MS lesion segmentation challenge\(^1\) and the ISBI 2015 longitudinal MS lesion segmentation

\(^1\)http://www.ia.unc.edu/MSseg/
challenger$^2$), which allows for a direct comparison with many state-of-the-art methods. In addition, we have used two much larger data sets containing two and four different MRI sequences from a multi-centre clinical trial in secondary progressive MS and relapsing remitting MS, which tends to have the most heterogeneity among the MS subtypes. These data sets are challenging due to the large variability in lesion size, shape, location, and intensity as well as varying contrasts produced by different scanners. The clinical trial data sets were used to estimate the number of images required to train a CEN without overfitting and to carry out a detailed analysis of different CEN architectures using different combinations of modalities, with a comparison to five publicly available state-of-the-art methods.

4.4.1 Data Sets and Preprocessing

Public Data Sets

The data set of the MICCAI 2008 MS lesion segmentation challenge (Styner et al., 2008) consists of 43 T1-weighted (T1w), T2-weighted (T2w), and fluid-attenuated inversion recovery (FLAIR) MRIs, divided into 20 training cases for which ground truth segmentations are made publicly available, and 23 test cases. In a first experiment, we evaluated our method on the 20 training cases using 5-fold cross-validation. In a second experiment, we trained our model on the 20 training cases and then used the trained model to segment the 23 test cases, which were sent to the challenge organizers for independent evaluation.

The data set of the ISBI 2015 longitudinal MS lesion segmentation challenge consists of 21 visit sets, each with T1w, T2w, proton density-weighted (PDw), and FLAIR MRIs. The challenge was not open for new submissions at the time of writing this article. Therefore,

$^2$http://iacl.ece.jhu.edu/MSChallenge
Table 4.1: Population, disease and scanner characteristics of the two clinical trial data sets.

<table>
<thead>
<tr>
<th></th>
<th>Data set 1</th>
<th>Data set 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MS subtype</strong></td>
<td>Secondary progressive MS</td>
<td>Relapsing remitting MS</td>
</tr>
<tr>
<td><strong>EDSS</strong></td>
<td>5.5 ± 1.0</td>
<td>3.3 ± 1.4</td>
</tr>
<tr>
<td><strong>Age</strong></td>
<td>49.58 ± 8.00</td>
<td>37.37 ± 8.81</td>
</tr>
<tr>
<td><strong>Gender</strong></td>
<td>male</td>
<td></td>
</tr>
<tr>
<td></td>
<td>175</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>female</td>
<td>311</td>
</tr>
<tr>
<td></td>
<td>unknown</td>
<td>14</td>
</tr>
<tr>
<td><strong>Field strength</strong></td>
<td>1 T</td>
<td></td>
</tr>
<tr>
<td></td>
<td>26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.5 T</td>
<td>383</td>
</tr>
<tr>
<td></td>
<td>3 T</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>unknown</td>
<td>14</td>
</tr>
</tbody>
</table>

we evaluated our method on the training set using leave-one-subject-out cross-validation, following the evaluation protocol of the second place method by Jesson and Arbel (2015) and third place method by Maier and Handels (2015) from the challenge proceedings. The paper of the first place method does not have sufficient details to replicate their evaluation.

**Clinical Trial Data Sets**

We used two different clinical trial data sets to evaluate our method (see Table 4.1). The first data set contains images of 500 secondary progressive MS patients split equally into training and test sets. The images were acquired from 45 different scanning sites. For each subject, the data set contains T2- and PD-weighted MRIs with a voxel size of 0.937 × 0.937 × 3.000 mm. The main preprocessing steps included rigid intra-subject registration, brain extraction, intensity normalization, and background cropping. This data set was used to evaluate how many images are required to train a simple 3-layer CEN without overfitting. Training on a single GeForce GTX 780 graphics card took between 6
and 32 hours per model depending on the training set size. However, once the network is trained, segmentation of new images can be performed in less than one second.

The second data set was collected from 67 different scanning sites using different 1.5 T and 3 T scanners, and consists of T1w, T2w, PDw, and FLAIR MRIs from 195 relapsing remitting MS patients, most with two time points (377 visit sets in total). In contrast to the previous data set, this data set also contains T1w and FLAIR MRIs, which allows for a direct comparison with other competing methods, which require these modalities. The image dimensions and voxel sizes vary by site, but most of the T1w images have close to 1 mm isotropic voxels, while the other images have voxel sizes close to $1 \times 1 \times 3$ mm. All images were skull-stripped using the brain extraction tool (BET) (Jenkinson et al., 2005), followed by an intensity normalization to the interval $[0, 1]$, and a 6 degree-of-freedom intra-subject registration using one of the 3 mm scans as the target image to align the different modalities. To speed-up the training, all images were cropped to a $164 \times 206 \times 52$ voxel subvolume with the brain roughly centred. The ground truth segmentations for both data sets were produced using an existing semiautomatic 2D region-growing technique, which has been used successfully in a number of large MS clinical trials (e.g., Kappos et al., 2006; Traboulsee et al., 2008). Each lesion was manually identified by an experienced radiologist and then interactively grown from the seed point by a trained technician.

We divided the data set into a training ($n = 250$), validation ($n = 50$), and test set ($n = 77$) such that images of each set were acquired from different scanning sites. The training, validation, and test sets were used for training our models, for monitoring the training progress, and to evaluate performance, respectively. The training set was also used to perform parameter tuning of the other methods used for comparison. Pre-training and fine-tuning of our 7-layer CEN-s took approximately 27 hours and 37 hours,
respectively, on a single GeForce GTX 780 graphics card. Once the network is trained, new multi-contrast images can be segmented in less than one second.

### 4.4.2 Comparison to Other Methods

We compared our method with five publicly available methods, some of which are widely used for clinical research and are established in the literature (e.g., Guizard et al., 2015; Subbanna et al., 2015; Sudre et al., 2015) as reference points for comparison. These five methods include:

1. Expectation maximization segmentation (EMS) method (Van Leemput et al., 2001);
2. Lesion growth algorithm (LST-LGA) (Schmidt et al., 2012), as implemented in the Lesion Segmentation Toolbox (LST) version 2.0.11;
3. Lesion prediction algorithm (LST-LPA) also implemented in the same LST toolbox;
4. Lesion-TOADS version 1.9 R (Shiue et al., 2010); and
5. Salem Lesion Segmentation (SLS) toolbox (Roura et al., 2015).

The Lesion-TOADS software only takes T1w and FLAIR MRIs and has no tunable parameters, so we used the default parameters to carry out the segmentations. The performance of EMS depends on the choice of the Mahalanobis distance $\kappa$, the threshold $t$ used to binarize the probabilistic segmentation, and the modalities used. We applied EMS to segment lesions using two combinations of modalities: a) T1w, T2w, and PDw, as used in the original paper (Van Leemput et al., 2001), and b) all four available modalities (T1w, T2w, PD2, FLAIR). We compared the segmentations produced for all combinations of $\kappa = 2.0, 2.2, \ldots, 4.6$ and $t = 0.05, 0.10, \ldots, 1.00$ with the ground truth segmentations on the
training set and chose the values that maximized the average DSC ($\kappa = 2.6, t = 0.75$ for three modalities; $\kappa = 2.8, t = 0.9$ for four modalities).

The LST-LGA and LST-LPA of the LST toolbox only take T1w and FLAIR MRIs as input, and we used those modalities to tune the initial threshold $\kappa$ of LST-LGA for $\kappa = 0.05, 0.10, \ldots, 1.00$ and the threshold $t$ used by LST-LPA to binarize the probabilistic segmentations for $t = 0.05, 0.10, \ldots, 1.00$. The optimal parameters were $\kappa = 0.10$ and $t = 0.45$, respectively.

Similarly, the SLS toolbox, which is the most recently published work (ROURA et al., 2015) also only takes T1w and FLAIR MRIs as input. This method uses an initial brain tissue segmentation obtained from the T1w images and segments lesions by treating lesional pixels as outliers to the normal appearing grey matter brain tissue on the FLAIR images. This method has three key parameters: $\omega_{\text{nb}}$, $\omega_{\text{ts}}$, and $\alpha_{\text{sls}}$. We tuned these parameters on the training set via a grid-search over a range of values as suggested by ROURA et al. (2015).

4.4.3 Measures of Segmentation Accuracy

We used the following six measures to produce a comprehensive evaluation of segmentation accuracy as there is generally no single measure that is sufficient to capture all information relevant to the quality of a produced segmentation (GARCÍA-LORENZO et al., 2013).

The first measure is the DSC (DICE, 1945) that computes a normalized overlap value between the produced and ground truth segmentations, and is defined as

$$\text{DSC} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}}, \quad (4.20)$$
where TP, FP, and FN denote the number of true positive, false positive, and false negative voxels, respectively. A value of 100% indicates a perfect overlap of the produced segmentation and the ground truth. The DSC incorporates measures of over- and underestimation into a single metric, which makes it a suitable measure to compare overall segmentation accuracy. In addition, we have used the true positive rate (TPR) and the positive predictive value (PPV) to provide further information on specific aspects of segmentation performance. The TPR is used to measure the fraction of the lesion regions in the ground truth that are correctly identified by an automatic method. It is defined as

$$\text{TPR} = \frac{TP}{TP + FN}, \quad (4.21)$$

where a value of 100% indicates that all true lesion voxels are correctly identified. The PPV is used to determine the extent of the regions falsely classified as lesion by an automatic method. It is defined as the fraction of true lesion voxels out of all identified lesion voxels

$$\text{PPV} = \frac{TP}{TP + FP}, \quad (4.22)$$

where a value of 100% indicates that all voxels that are classified as lesion voxels are indeed lesion voxels as defined by the ground truth (no false positives).

We also measured the relative absolute volume difference (VD) between the ground truth and the produced segmentation by computing their volumes (Vol), i.e.,

$$\text{VD} = \frac{\text{Vol(Seg)} - \text{Vol(GT)}}{\text{Vol(GT)}}, \quad (4.23)$$

where Seg and GT denote the obtained segmentation and ground truth, respectively. However, it has been noted (GARCÍA-LORENZO et al., 2013) that a wide variability exists
even between the lesion segmentations of trained experts, and thus, the achieved volume differences reported in the literature have ranged from 10% to 68%.

For more precise evaluation, we have also included the lesion-wise true positive rate (LTPR) and the lesion-wise false positive rate (LFPR) that are much more sensitive in measuring the segmentation accuracy of smaller lesions, which are important to detect when performing early disease diagnosis (García-Lorenzo et al., 2013). More specifically, the LTPR measures the TPR on a per lesion-basis and is defined as

$$\text{LTPR} = \frac{\text{LTP}}{\#RL},$$  \hspace{1cm} (4.24)

where LTP denotes the number of lesion true positives, i.e., the number of lesions in the reference segmentation that overlap with a lesion in the produced segmentation, and #RL denotes the total number of lesions in the reference segmentation. An LTPR with a value of 100% indicates that all lesions are correctly identified. Similarly, the lesion-wise false positive rate (FPR) measures the fraction of the segmented lesions that are not in the ground truth and is defined as

$$\text{LFPR} = \frac{\text{LFP}}{\#PL},$$  \hspace{1cm} (4.25)

where LFP denotes the number of lesion false positives, i.e., the number of lesions in the produced segmentation that do not overlap with a lesion in the reference segmentation, and #PL denotes the total number of lesions in the produced segmentation. An LFPR with a value of 0% indicates that no lesions were incorrectly identified.
4.4.4 Training Parameters

In this set of experiments, we evaluated the impact of the number of epochs and the sensitivity ratio on the trained networks. Figure 4.2 shows the mean DSC evaluated on the training and validation sets of the second clinical data set as computed during training of a 7-layer CEN-s up to 500 epochs. The mean DSC scores increase monotonically, but the improvements are minor after 400 epochs. The optimal number of epochs is a trade-off between accuracy and time required for training. Due to the relatively small improvements after 400 epochs, we decided to stop the training procedure at 500 epochs. For the challenge data sets, due to their small sizes, we did not employ a subset of the data for a dedicated validation set to choose the number of epochs. Instead, we set the number of epochs to 2500, which corresponds to roughly the same number of gradient updates compared to the clinical trial data set.

To determine an effective sensitivity ratio, we measured the performance on the validation set over a range of values. For each choice of ratio, we binarized the segmentations
using a threshold that maximized the DSC on the training set. Figure 4.3 shows a set of receiver operating characteristic (ROC) curves for different choices of the sensitivity ratio ranging from 0.01 to 0.10 and the corresponding optimal thresholds. The plots illustrate our findings that our method is not sensitive to the choice of the sensitivity ratio, which mostly affects the optimal threshold. We chose a fixed sensitivity ratio of 0.02 for all our experiments.

4.4.5 Impact of the Training Set Size on the Segmentation Performance

To evaluate the impact of the training set size on the segmentation performance, we trained a CEN with 3 layers on the first in-house data set with varying number of training samples and calculated the mean DSC on the training and test sets as illustrated in Figure 4.4. For small training sets, there is a large difference between the DSCs on the training and test sets, which indicates that the training set is too small to learn a representative set of

![Figure 4.3: ROC curves for different sensitivity ratios $r$. A ‘+’ marks the TPR and FPR of the optimal threshold. Varying the value of $r$ results in almost identical ROC curves and only causes a change of the optimal threshold $t$, which shows the robustness of our method with respect to the sensitivity ratio.](image)
features. At around 100 samples, the model becomes stable in terms of test performance and the small differences between training and test DSCs, indicating that overfitting of the training data is no longer occurring. With 100 training subjects, our method achieves a mean DSC on the test set of 57.38%.

4.4.6 Comparison on Public Data Sets

To allow for a direct comparison with a large number of state-of-the-art methods, we evaluated our method on the MICCAI 2008 MS lesion segmentation challenge \cite{Styner2008} and the ISBI 2015 longitudinal MS lesion segmentation challenge. As shown in the previous section, approximately 100 images are required to train the 3-layer CEN without overfitting and we expect the required number of images to be even higher when adding more layers. Due to the relatively small size of the training data sets provided by
the two challenges, we used a CEN with only three layers on these data sets to reduce the risk of overfitting. The parameters of the models are summarized in Table 4.2.

**MICCAI 2008 MS Lesion Segmentation Challenge**

In a first experiment, we evaluated the performance of our method using 5-fold cross-validation on the training set of the MICCAI challenge. Figure 4.5 shows a comparison of lesion masks produced by our method with the ground truth for three subjects. The first two rows show the FLAIR, T1w, T2w, ground truth segmentations, and predicted segmentations of two subjects with a DSC of 60.58% and 61.37%. Despite the large contrast differences between the two subjects, our method performed well and consistently, which indicates that our model was able to learn features that are robust to a large range of intensity variations. The last row shows a subject with a DSC of 9.01%, one of the lowest DSC scores from the data set. Our method segmented lesions that have similar contrast to the other two subjects, but these regions were not classified as lesions by the manual rater. This highlights the difficulty of manual lesion segmentation, as the difference between diffuse white matter pathology and focal lesions is often indistinct.

A quantitative comparison of our method with other state-of-the-art methods is summarized in Table 4.3. Our method outperforms the winning method (Souplet et al., 2008) of the MS lesion segmentation challenge 2008 and the currently best unsupervised method.
Figure 4.5: Example segmentations of our method for three different subjects from the MICCAI challenge data set. Our method performed well and consistently despite the large contrast differences seen between the first two rows. In the third row, our method also segmented lesions that have similar contrast, but these regions had not been identified as lesions by the manual rater, which highlights the difficulty in distinguishing focal lesions from diffuse damage, even for experts.

reported on that data set (Weiss et al., 2013) in terms of mean TPR and PPV. Our method performs comparably to a current method (Geremia et al., 2011, 2010) that uses a carefully designed set of features specifically designed for lesion segmentation, despite our method having learned its features solely from a relatively small training set.

A comparison of our method with other state-of-the-art methods evaluated on the test set of the MICCAI challenge is summarized in Table 4.4. Our method ranked 6th (2nd if only considering methods with only one submission, i.e., without subsequent parameter-tuning and adjustments) out of 52 entries submitted to the challenge, outperforming
Table 4.3: Comparison of our method with state-of-the-art lesion segmentation methods in terms of mean TPR, PPV, and DSC on the training set of the MICCAI 2008 lesion segmentation challenge. Our method performs comparably to the best methods reported on the MS lesion segmentation challenge data set.

<table>
<thead>
<tr>
<th>Method</th>
<th>TPR</th>
<th>PPV</th>
<th>DSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOUPLET et al. (2008)</td>
<td>20.65</td>
<td>30.00</td>
<td>—</td>
</tr>
<tr>
<td>WEISS et al. (2013)</td>
<td>33.00</td>
<td>36.85</td>
<td>29.05</td>
</tr>
<tr>
<td>GEREMIA et al. (2010)</td>
<td>39.85</td>
<td>40.35</td>
<td>—</td>
</tr>
<tr>
<td>Our method</td>
<td>39.71</td>
<td>41.38</td>
<td>35.52</td>
</tr>
</tbody>
</table>

Table 4.4: Selected methods out of the 52 entries submitted for evaluation to the MICCAI 2008 MS lesion segmentation challenge. Columns LTPR, LFPR, and VD show the average computed from the two raters in percent. Challenge results last updated: Dec 15, 2015.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Method</th>
<th>Score</th>
<th>LTPR</th>
<th>LFPR</th>
<th>VD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,3,9</td>
<td>JESSON and ARBEL (2015)</td>
<td>86.94</td>
<td>48.7</td>
<td>28.3</td>
<td>80.2</td>
</tr>
<tr>
<td>2</td>
<td>GUIZARD et al. (2015)</td>
<td>86.11</td>
<td>49.9</td>
<td>42.8</td>
<td>48.8</td>
</tr>
<tr>
<td>4,20,26</td>
<td>TOMAS-FERNANDEZ and WARFIELD (2015)</td>
<td>84.46</td>
<td>46.9</td>
<td>44.6</td>
<td>45.6</td>
</tr>
<tr>
<td>5,7</td>
<td>JERMAN et al. (2015)</td>
<td>84.16</td>
<td>65.2</td>
<td>63.8</td>
<td>77.5</td>
</tr>
<tr>
<td>6</td>
<td>Our method</td>
<td>84.07</td>
<td>51.6</td>
<td>51.3</td>
<td>57.8</td>
</tr>
<tr>
<td>11</td>
<td>ROURA et al. (2015)</td>
<td>82.34</td>
<td>50.2</td>
<td>41.9</td>
<td>111.6</td>
</tr>
<tr>
<td>13</td>
<td>GEREMIA et al. (2010)</td>
<td>82.07</td>
<td>55.1</td>
<td>74.1</td>
<td>48.9</td>
</tr>
<tr>
<td>24</td>
<td>SHIEE et al. (2010)</td>
<td>79.90</td>
<td>52.4</td>
<td>72.7</td>
<td>74.5</td>
</tr>
</tbody>
</table>

the recent SLS by ROURA et al. (2015), and popular methods such as the random forest approach by GEREMIA et al. (2010), and Lesion-TOADS by SHIEE et al. (2010), but not as well as the patch-based segmentation approach by GUIZARD et al. (2015), or the model of population and subject (MOPS) approach by TOMAS-FERNANDEZ and WARFIELD (2015), which used additional images to build the intensity model of a healthy population. This is a very promising result for the first submission of our method given the simplicity of the model and the small training set size.
Table 4.5: Comparison of our method with the second and third ranked methods from the ISBI MS lesion segmentation challenge. The evaluation was performed on the training set using leave-one-subject-out cross-validation. GT1 and GT2 denote that the model was trained with the segmentations provided by the first and second rater as the ground truth, respectively.

<table>
<thead>
<tr>
<th>Method</th>
<th>Rater 1 DSC</th>
<th>LTPR</th>
<th>LFPR</th>
<th>Rater 1 DSC</th>
<th>LTPR</th>
<th>LFPR</th>
<th>Rater 2 DSC</th>
<th>LTPR</th>
<th>LFPR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rater 1</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>73.2</td>
<td>64.5</td>
<td>17.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rater 2</td>
<td>73.2</td>
<td>82.6</td>
<td>35.5</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JESSON and ARBEL (2015)</td>
<td>70.4</td>
<td>61.1</td>
<td>13.5</td>
<td>68.1</td>
<td>50.1</td>
<td>12.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAIER and HANDELS (2015) (GT1)</td>
<td>70</td>
<td>53</td>
<td>48</td>
<td>65</td>
<td>37</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAIER and HANDELS (2015) (GT2)</td>
<td>70</td>
<td>55</td>
<td>48</td>
<td>65</td>
<td>38</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Our method (GT1)</td>
<td>68.4</td>
<td>74.5</td>
<td>54.5</td>
<td>64.4</td>
<td>63</td>
<td>52.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Our method (GT2)</td>
<td>68.3</td>
<td>78</td>
<td>64.5</td>
<td>65.8</td>
<td>69.3</td>
<td>61.9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ISBI 2015 Longitudinal MS Lesion Segmentation Challenge

In addition, we evaluated our method on the 21 publicly available labeled cases from the ISBI 2015 longitudinal MS lesion segmentation challenge. The challenge organizers have only released the names of the top three teams, only two of which have published a summary of their mean DSC, LTPR, and LFPR scores for both raters to allow for a direct comparison. Following the evaluation protocol of the second (JESSON and ARBEL, 2015) and third (MAIER and HANDELS, 2015) place methods, we trained our model using leave-one-subject-out cross-validation on the training images and compared our results to the segmentations provided by both raters. Table 4.5 summarizes the performance of our method, the two other methods for comparison, and the performance of the two raters when compared against each other. Compared to the second and third place methods, our method was more sensitive and produced significantly higher LTPR scores, but also had more false positives, which resulted in slightly lower but still comparable DSC scores. This is again a promising result on a public data set.
Table 4.6: Parameters of the 3-layer CEN used on the clinical trial data set.

<table>
<thead>
<tr>
<th>Layer type</th>
<th>Kernel Size</th>
<th>#Filters</th>
<th>Number of units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>—</td>
<td>—</td>
<td>164 × 206 × 52 × 2</td>
</tr>
<tr>
<td>Convolutional</td>
<td>9 × 9 × 5 × 2</td>
<td>32</td>
<td>156 × 198 × 48 × 32</td>
</tr>
<tr>
<td>Deconvolutional</td>
<td>9 × 9 × 5 × 32</td>
<td>1</td>
<td>164 × 206 × 52 × 1</td>
</tr>
</tbody>
</table>

4.4.7 Comparison of Network Architectures, Input Modalities, and Publicly Available Methods on Clinical Trial Data

Quantitative Comparison

To determine the effect of network architectures, we compared the segmentation performance of three different networks using T1w and FLAIR MRIs. Specifically, we trained a 3-layer CEN and two 7-layer CENs, one with shortcut connections and one without. To investigate the effect of different input image types, we additionally trained two 7-layer CEN-s on the modalities used by EMS (T1w, T2w, PDw) and all four modalities (T1w, T2w, PDw, FLAIR). The parameters of the networks are given in Table 4.6 and Table 4.7. To roughly compensate for the anisotropic voxel size of the input images, we chose an anisotropic filter size of 9 × 9 × 5. In addition, we ran the five competing methods discussed in Section 4.4.2 with Lesion-TOADS, SLS, and the two LST methods using the T1w and FLAIR images, and EMS using three (T1w, T2w, PDw) and all four modalities in separate tests. A comparison of the segmentation accuracy of the trained networks and competing methods is summarized in Table 4.8.

All CEN architectures performed significantly better than all other methods regardless of the input modalities, with LST-LGA being the closest in overall segmentation accuracy. Comparing CEN to LST-LGA, the improvements in the mean DSC scores ranged from 3 percentage points (pp) for the 3-layer CEN to 17 pp for the 7-layer CEN with shortcut
Table 4.7: Parameters of the 7-layer CEN-s used on the clinical trial data set.

<table>
<thead>
<tr>
<th>Layer type</th>
<th>Kernel Size</th>
<th>#Filters</th>
<th>Number of units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>—</td>
<td>—</td>
<td>164 × 206 × 52 × 2</td>
</tr>
<tr>
<td>Convolutional</td>
<td>9 × 9 × 5 × 2</td>
<td>32</td>
<td>156 × 198 × 48 × 32</td>
</tr>
<tr>
<td>Average Pooling</td>
<td>2 × 2 × 2</td>
<td>—</td>
<td>78 × 99 × 24 × 32</td>
</tr>
<tr>
<td>Convolutional</td>
<td>9 × 10 × 5 × 32</td>
<td>32</td>
<td>70 × 90 × 20 × 32</td>
</tr>
<tr>
<td>Deconvolutional</td>
<td>9 × 10 × 5 × 32</td>
<td>32</td>
<td>78 × 99 × 24 × 32</td>
</tr>
<tr>
<td>Unpooling</td>
<td>2 × 2 × 2</td>
<td>—</td>
<td>156 × 198 × 48 × 32</td>
</tr>
<tr>
<td>Deconvolutional</td>
<td>9 × 9 × 5 × 32</td>
<td>1</td>
<td>164 × 206 × 52 × 1</td>
</tr>
</tbody>
</table>

trained on all four modalities. The improved segmentation performance was mostly due to an increase in lesion sensitivity. LST-LGA achieved a mean lesion TPR of 37.50 %, compared to 54.55 % produced by the CEN with shortcut when trained on the same modalities, and 62.49 % when trained on all four modalities, while achieving a comparable number of lesion false positives. The mean lesion FPRs and mean volume differences of LST-LGA and the 7-layer CEN-s were very close, when trained on the same modalities, and the CEN-s further reduced its FPR when trained on more modalities.

This experiment also showed that increasing the depth of the CEN and adding the shortcut connections both improve the segmentation accuracy. Increasing the depth of the CEN from three layers to seven layers improved the mean DSC by 3 pp. The improvement was confirmed to be statistically significant using a one-sided paired \( t \)-test (\( p \)-value of \( 1.3 \times 10^{-5} \)). Adding a shortcut to the network further improved the segmentation accuracy as measured by the DSC by 3 pp. A second one-sided paired \( t \)-test was performed to confirm the statistical significance of the improvement with a \( p \)-value of less than \( 1 \times 10^{-10} \).

Qualitative Comparison of Network Architectures

The impact of increasing the depth of the network on the segmentation performance of very large lesions is illustrated in Figure 4.6, where the true positive, false negative, and
Table 4.8: Comparison of the segmentation accuracy of different CEN models, other methods, and input modalities. The table shows the mean of the Dice similarity coefficient (DSC), lesion true positive rate (LTPR), and lesion false positive rate (LFPR). Because the volume difference (VD) is not limited to the interval $[0, 100]$, a single outlier can heavily affect the calculation of the mean. We therefore excluded outliers before calculating the mean of the VD for all methods using the box plot criterion.

<table>
<thead>
<tr>
<th>Method</th>
<th>DSC [%]</th>
<th>LTPR [%]</th>
<th>LFPR [%]</th>
<th>VD [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input modalities: T1w and FLAIR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-layer CEN</td>
<td>49.24</td>
<td>57.33</td>
<td>61.39</td>
<td>43.45</td>
</tr>
<tr>
<td>7-layer CEN</td>
<td>52.07</td>
<td>43.88</td>
<td>29.06</td>
<td>37.01</td>
</tr>
<tr>
<td>7-layer CEN-s</td>
<td>55.76</td>
<td>54.55</td>
<td>38.64</td>
<td>36.30</td>
</tr>
<tr>
<td>Lesion-TOADS (Shiee et al., 2010)</td>
<td>40.04</td>
<td>56.56</td>
<td>82.90</td>
<td>49.36</td>
</tr>
<tr>
<td>SLS (Roura et al., 2015)</td>
<td>43.20</td>
<td>56.80</td>
<td>50.80</td>
<td>12.30</td>
</tr>
<tr>
<td>LST-LGA (Schmidt et al., 2012)</td>
<td>46.64</td>
<td>37.50</td>
<td>38.06</td>
<td>36.77</td>
</tr>
<tr>
<td>LST-LPA (Schmidt et al., 2012)</td>
<td>46.07</td>
<td>48.02</td>
<td>52.94</td>
<td>41.62</td>
</tr>
<tr>
<td><strong>Input modalities: T1w, T2w, and PDw</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7-layer CEN-s</td>
<td>61.18</td>
<td>52.00</td>
<td>36.68</td>
<td>29.38</td>
</tr>
<tr>
<td>EMS (Van Leemput et al., 2001)</td>
<td>42.94</td>
<td>44.80</td>
<td>76.58</td>
<td>49.29</td>
</tr>
<tr>
<td><strong>Input modalities: T1w, T2w, FLAIR, and PDw</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7-layer CEN-s</td>
<td>63.83</td>
<td>62.49</td>
<td>36.10</td>
<td>32.89</td>
</tr>
<tr>
<td>EMS (Van Leemput et al., 2001)</td>
<td>39.70</td>
<td>49.08</td>
<td>85.01</td>
<td>34.51</td>
</tr>
</tbody>
</table>

false positive voxels are highlighted in green, yellow, and red, respectively. The receptive field of the 3-layer CEN has a size of only $17 \times 17 \times 9$ voxels, which reduces its ability to identify very large lesions marked by two white circles. In contrast, the 7-layer CEN has a receptive field size of $49 \times 53 \times 26$ voxels, which allows it to learn features that can capture much larger lesions. Consequently, the 7-layer CEN, with and without shortcut, is able to learn a feature set that captures large lesions much better than the 3-layer CEN, which results in an improved segmentation. However, increasing the depth of the network without adding shortcut connections reduces the network’s sensitivity to very small lesions as illustrated in Figure 4.7. In this example, the 3-layer CEN was able to detect three small lesions, indicated by the white circles, which were missed by the 7-layer CEN. Adding
Table 4.9: Lesion size groups as used for the detailed analysis.

<table>
<thead>
<tr>
<th>Group</th>
<th>Mean lesion size [mm$^3$]</th>
<th>#Samples</th>
<th>Lesion load [mm$^3$]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very small</td>
<td>[0, 70]</td>
<td>6</td>
<td>1457 ± 1492</td>
</tr>
<tr>
<td>Small</td>
<td>(70, 140]</td>
<td>24</td>
<td>4298 ± 2683</td>
</tr>
<tr>
<td>Medium</td>
<td>(140, 280]</td>
<td>24</td>
<td>12620 ± 9991</td>
</tr>
<tr>
<td>Large</td>
<td>(280, 500]</td>
<td>14</td>
<td>13872 ± 5814</td>
</tr>
<tr>
<td>Very large</td>
<td>&gt; 500</td>
<td>9</td>
<td>35238 ± 27531</td>
</tr>
</tbody>
</table>

shortcut connections enables our model to learn a feature set that spans a wider range of lesion sizes, which increases the sensitivity to small lesions and, hence, allows the 7-layer CEN-s to detect all three small lesions (highlighted by the white circles), while still being able to segment large lesions.

4.4.8 Comparison for Different Lesion Sizes

To examine the effect of lesion size on segmentation performance, we stratified the test set into five groups based on their mean reference lesion size as summarized in Table 4.9. A comparison of segmentation accuracy and lesion detection measures of a 7-layer CEN-s trained on different input modalities and the best performing competing method LST-LGA for different lesion sizes is illustrated in Figure 4.8. The 7-layer CEN-s outperformed LST-LGA for all lesions sizes except for very large lesions when trained on T1w and FLAIR MRIs. The advantage extended to all lesion sizes when the CEN-s was trained on all four modalities, which could not be done for LST-LGA. The differences were larger for smaller lesions, which are generally more challenging to segment for all methods. The differences between the two approaches were due to a higher sensitivity to lesions as measured by the LTPR, especially for smaller lesions, while the number of false positives was approximately the same for all lesion sizes.
Figure 4.6: Impact of increasing the depth of the network on the segmentation performance of very large lesions. The true positive, false negative, and false positive voxels are highlighted in green, yellow, and red, respectively. The 7-layer CEN, with and without shortcut, is able to segment large lesions much better than the 3-layer CEN due to the increased size of the receptive field. This figure is best viewed in colour.

Figure 4.7: Comparison of segmentation performance of different CEN architectures for small lesions. The white circles indicate lesions that were detected by the 3-layer CEN and the 7-layer CEN, but only with shortcut. Increasing the network depth decreases the sensitivity to small lesions, but the addition of a shortcut allows the network to regain this ability, while still being able to detect large lesions (see Figure 4.6). This figure is best viewed in colour.
Figure 4.8: Comparison of segmentation accuracy and lesion detection measures of a 7-layer CEN-s trained on different input modalities and the best performing competing method LST-LGA for different lesion sizes. The 7-layer CEN-s outperforms LST-LGA for all lesions sizes except for very large lesions when trained on T1w and FLAIR MRIs, and for all lesion sizes when trained on all four modalities, due to a higher sensitivity to lesions, while producing approximately the same number of false positives. Outliers are denoted by black dots.
4.5 Discussion

The automatic segmentation of MS lesions is a very challenging task due to the large variability in lesion size, shape, intensity, and location, as well as the large variability of imaging contrasts produced by different scanners used in multi-centre studies. Most unsupervised methods model lesions as an outlier class or a separate cluster in a subject-specific model, which makes them inherently robust to inter-subject and inter-scanner variability. However, outliers are often not specific to lesions and can also be caused by intensity inhomogeneities, partial volume, imaging artifacts, and small anatomical structures such as blood vessels, which leads to the generation of false positives. On the other hand, supervised methods can learn to discriminate between lesion and non-lesion tissue, but are more sensitive to the variability in lesion appearance and different contrasts produced by different scanners. To overcome those challenges, supervised methods require large data sets that span the variability in lesion appearance and careful preprocessing to match the imaging contrast of new images with those of the training set. Library-based approaches have shown great promise for the segmentation of MS lesions, but do not scale well to very large data sets due to the large amount of memory required to store comprehensive sample libraries and the time required to scan such libraries for matching patches. On the other hand, parametric deep learning models such as convolutional neural networks scale much better to large training sets, because the size required to store the model is independent of the training set size, and the operations required for training and inference are inherently parallelizable, which allows them to take advantage of very fast GPU-accelerated computing hardware. Furthermore, the combination of many nonlinear processing units allows them to learn features that are robust under large variability, which is crucial for the segmentation of MS lesions.
Convolutional neural networks were originally designed to classify entire images and designing networks that can segment images remains an important research topic. Early approaches have formulated the segmentation problem as a patch-wise classification problem, which allows them to directly use established classification network architectures for image segmentation. However, a major limitation of patch-based deep learning approaches is the time required for training and inference. Fully convolutional networks can perform the segmentation much more efficiently, but generally lack the precision to perform voxel-accurate segmentation and cannot handle unbalanced classes.

To overcome these challenges, we have presented a new method for the automatic segmentation of MS lesions based on deep convolutional encoder networks with shortcut connections. The joint training of the feature extraction and prediction pathways allows for the automatic learning of features at different scales that are tuned for a given combination of image types and segmentation task. Shortcuts between the two pathways allow high- and low-level features to be leveraged at the same time for more consistent performance across scales. In addition, we have proposed a new objective function based on the combination of sensitivity and specificity, which makes the objective function inherently robust to unbalanced classes such as MS lesions, which typically comprise less than 1% of all image voxels. We have evaluated our method on two publicly available data sets and a large data set from an MS clinical trial, with the results showing that our method performs comparably to the best state-of-the-art methods, even for relatively small training set sizes. We have also shown that when a suitably large training set is available, our method is able to segment MS more accurately than widely-used competing methods such as EMS, LST-LGA, SLS, and Lesion-TOADS. The substantial gains in accuracy were mostly due to an increase in lesion sensitivity, especially for small lesions. Overall, our proposed CEN with shortcut connections performed consistently well over a wide range of lesion sizes.
Our segmentation framework is very flexible and can be easily extended. One such extension could be to incorporate prior knowledge about the tissue type of each non-lesion voxel into the segmentation procedure. The probabilities of each tissue class could be precomputed by a standard segmentation method, after which they can be added as an additional channel to the input units of the CEN, which would allow the CEN to take advantage of intensity information from different modalities and prior knowledge about each tissue class to carry out the segmentation. In addition, our method can be applied to other segmentation tasks. Although we have only focused on the segmentation of MS lesions in this paper, our method does not make any assumptions specific to MS lesion segmentation. The features required to carry out the segmentation are solely learned from training data, which allows our method to be used to segment different types of pathology or anatomy when a suitable training set is available.
Chapter 5

Manifold Learning by Deep Learning

5.1 Introduction

Changes in brain morphology such as global and regional atrophy and the formation of white matter lesions are two hallmarks of multiple sclerosis (MS) pathology. In order to quantify the pathological manifestation of MS, a number of imaging biomarkers derived from magnetic resonance imaging (MRI) scans of the brain, such as lesion volume and whole brain volume, have been proposed, which have established their importance for the study of MS. However, MS is a complex disease whose pathological variability extends well beyond what can be captured by global and local volumetric measures. It would be highly desirable to have a method that can automatically discover potentially important patterns of variability in brain morphology and lesion distribution, which would advance our understanding of the complex pathology of MS. In addition, the joint modelling of brain morphology and lesion distribution would further our knowledge of how these two key pathological features interact. However, this type of modelling is very challenging due to the high dimensionality of the data.
In recent years, there has been an increased interest in biomarker discovery using manifold learning to form high-level, low-dimensional representations of medical images (Aljabar et al., 2011; Wolz et al., 2012, 2010b). In order to discover common patterns of variability in a group of images, each image of the data set is regarded as a point in a high-dimensional image space (called the ambient space), with \( n_x \times n_y \times n_z \) coordinates, where \( n_x, n_y, n_z \) are the dimensions of each image. On the other hand, each image could also be identified by a smaller set of parameters that describe shape variations and patterns that are common for a particular group of images. These parameters span a new space called the manifold space. The task of manifold learning is to discover the low-dimensional space and its parameters, which can then be used to model the anatomical variability within a population, or serve as biomarkers to track disease state and progression.

In the remainder of this section, we will briefly revise common manifold learning techniques and applications for medical image analysis. A main part of the chapter will be an explanation of the deep learning manifold learning framework using a deep belief network. We have evaluated the proposed manifold learning method on two tasks: a) to learn the variability of magnetic resonance (MR) images of the brain of Alzheimer’s disease (AD) patients, and b) to discover patterns of morphological variability and lesion distribution in MS.

### 5.2 Manifold Learning for Medical Image Analysis

Most popular manifold learning methods for medical image analysis can be roughly classified into local and global methods (Cayton, 2005). Local algorithms such as locally linear embedding (LLE) (Saul and Roweis, 2003) and Laplacian eigenmaps (LEM) (Belkin and Niyogi, 2003) try to find a low dimensional representation of the data such that
local distances between points in ambient space are best preserved when mapped to manifold space. In contrast, global methods such as isometric feature mapping (Isomap) (Tenenbaum et al., 2000) try to find an embedding of the manifold space that best preserves geodesic distances of points that lie far apart in ambient space. Both types of methods require building a proximity graph that is used to capture neighborhood information (local methods) or to estimate geodesic distances by calculating the shortest path distances between all points on the affinity graph (global methods). Both types of methods assume that the manifold is locally linear, which means that distances between neighboring points in manifold space can be approximated by their distances in ambient space. Once the proximity graph has been created, a unique solution to the inherent optimization problem of each algorithm can be found analytically by solving an eigenvalue problem.

Manifold learning methods have been successfully applied to various image analysis problems such as the segmentation of the hippocampus (Wolz et al., 2010a), to regularize the segmentation of heart (Zhang et al., 2006) and brain ventricles (Etyngier et al., 2007), and to constrain the deformable registration of brain images to have biologically plausible parameters (Hamm et al., 2010), while most methods focus on clinical prediction (Aljabar et al., 2011; Bhatia et al., 2012; Duchateau et al., 2012; Gerber et al., 2010; Guerrero et al., 2014; Wachinger et al., 2015; Wolz et al., 2012). Gerber et al. used Isomap to predict clinical parameters of AD patients (Gerber et al., 2010, 2009), and Wolz et al. used LEM to perform biomarker discovery (Wolz et al., 2011, 2012), also of AD patients. Duchateau et al. (2011, 2012) discriminate between normal and abnormal motion patterns of the heart based on the definition of pathology as a deviation from normality. After learning the manifold representing normal motion patterns, abnormal motion patterns can be detected based on the distance of a new data sample from the manifold surface. Wachinger et al. (2015) proposed a new shape descriptor of the brain called BrainPrint, which is
build by solving the eigenvalue problem of the 2D and 3D Laplace-Beltrami operator on meshes of segmented cortical and subcortical structures. The descriptor defines a similarity measure on brain images, which can be used to perform subject identification (Wachinger et al., 2014b), and to predict Alzheimer’s disease (Wachinger et al., 2014a). Bhatia et al. (2012) model regional variations of the brain by building a manifold on hierarchical image patches and applied it to finding discriminative regions of 3D brain MR images for classifying Alzheimer’s disease. Guerrero et al. (2014) have used manifold learning on regions of interest (ROIs) to extract features that represent inter-subject variability of AD patients. In a first step, sparse regression is used to automatically detect the ROI using mini-mental state examination (MMSE) scores as the independent variable. Then, LEM with cross-correlation as the distance measure is used to learn the manifold of the ROIs, from which the features are extracted.

A main challenge of most manifold learning methods (e.g., Isomap, LEM, LLE) is the construction of the proximity graph. Building the proximity graph requires choosing an appropriate neighborhood criterion, which can be challenging due to the sensitivity to noise of commonly used criteria (e.g., $k$-nearest neighbors, $\epsilon$-ball), and, consequently, may result in an unstable topology of the learned manifold (Balasubramanian and Schwartz, 2002). Other challenges include avoiding erroneously disconnected regions and shortcuts, finding a trade-off between sampling density and curvature of local patches of the manifold, and finding a suitable distance metric. A number of solutions have been proposed, but there is no general solution. To increase the robustness to noise and varying parameter settings, Carreira-Perpiñán and Zemel (2005) proposed building the graph from a graph ensemble that combines multiple spanning trees, each fit to a perturbed version of the data set. A trade-off between sampling density and curvature of local patches of the manifold can be found automatically by adapting the selection of the
neighborhood sizes through neighborhood contraction and expansion (Zhang et al., 2012). Gerber et al. (2010) have shown that the choice of a suitable distance measure is crucial for manifold learning using Isomap and that the warping distance between brain images improves the learning performance over previously used Euclidean distances in the image space.

5.3 Manifold Learning using Deep Belief Networks

Deep generative models such as deep belief networks (DBNs) (Hinton et al., 2006) are a promising alternative to previously used brain manifold learning methods due to their ability to discover patterns of similarity in groups of images. In contrast to most commonly used manifold learning algorithms (e.g., LLE, LEM, Isomap), DBNs do not assume the manifold space to be locally linear and do not require a previously defined similarity measure or the construction of a proximity graph, which makes them more generally applicable to a wide range of manifold learning tasks. This is particularly important for the application to brain manifold learning, because the high-dimensionality of brain MRIs make defining a neighborhood criterion challenging.

5.3.1 General Architecture

The general architecture of the DBN used for manifold learning of brain MRIs is illustrated in Figure 5.1. The first restricted Boltzmann Machine (RBM) (Freund and Haussler, 1992; Hinton, 2010) receives the intensity values of a group of images in ambient space as input and reduces the dimensionality of each image by discovering patterns of similarity that are common within groups of images. Subsequent RBMs receive the hidden unit activations of the previous RBM as input, thus learning successively more complex and
abstract patterns from a training set, where the hidden units of the last layer represent the coordinates of an image in manifold space. The number of trainable weights increases significantly with the resolution of the training images. In order to scale the model to high-resolution images, the first several layers of our DBN are strided convolutional restricted Boltzmann machines (sconvRBMs) (Brosch and Tam, 2015), a type of RBM that uses weight sharing and local connectivity between units of adjacent layers to reduce the number of trainable weights. Due to the much smaller number of trainable parameters compared to dense RBMs, sconvRBMs are best suited for learning low- to mid-level features from very high-dimensional data such as brain MRIs. Compared to other more commonly used convolution-based RBMs (Lee et al., 2009), an advantage of sconvRBMs is that inference is invertible, which allows the reconstruction of the visible units from the hidden unit activations. In our application, this allows for the mapping of parameters from the manifold space back to the ambient space. Due to the local connectivity of the weights, sconvRBMs can only learn local patterns in the data. In order to learn patterns that describe images as a whole, sconvRBM layers are followed by dense RBMs, which integrate the extracted feature from all image locations in order to model global patterns.

5.3.2 Unit Types

To apply the model to real-valued data like the intensities of some medical images, the visible units are modelled as Gaussian units. As suggested in Hinton et al.’s RBM training guide (Hinton, 2010), we did not learn the variance of the Gaussian units and standardized the inputs to having zero mean and unit variance instead. There are two typical choices for the hidden units of an RBM, binary hidden units and noisy rectified linear units (NReLUs). A binary hidden unit can only encode two states. In order to learn patterns of variability that span a continuous spectrum of changes instead of binary on/off
patterns, we use NReLU units as the hidden units, which have also shown to improve the learning performance (Nair and Hinton, 2010) of RBMs.

5.3.3 Incorporating a Region of Interest

A challenge for training DBNs on brain MRIs is that large black regions without local structure can lead to random activations of the hidden units and consequently to the learning of random filters. To overcome this problem, we propose incorporating a ROI term into the energy equation of the convolutional restricted Boltzmann machine (convRBM), which allows constraining the filter learning process to a given ROI. This can be achieved by the element-wise multiplication of the visible and hidden units with a binary mask,
which sets the visible and hidden units outside of the ROI to zero, thereby removing their contribution to the energy of the model. The modified energy of the model is given by

$$
E(v, h, r_v, r_h) = - \sum_{i=1}^{N_c} \sum_{j=1}^{N_k} (r_h \cdot h^{(j)}) \bullet (\tilde{w}^{(ij)} \ast (r_v \cdot v^{(i)})) \\
- \sum_{i=1}^{N_c} b_i \sum_{x, y=1}^{N_v} r_{v, xy} v^{(i)}_{xy} - \sum_{j=1}^{N_k} c_j \sum_{x, y=1}^{N_h} r_{h, xy} h^{(j)}_{xy},
$$

(5.1)

where $r_v$ and $r_h$ are binary masks defining the ROI with respect to the visible and hidden units, and $\cdot$ denotes element-wise multiplication. Because sconvRBM can be mapped to equivalent convRBM, we can use the same modification to restrict the learning of filters of sconvRBM.

### 5.4 Manifold of MRIs of Alzheimer’s Disease Patients

In a first experiment, we applied our manifold learning framework using DBNs to the task of learning the manifold of brain MRIs of AD and healthy subjects. DBNs have shown to be able to learn the manifold of handwritten digits (Hinton et al., 2006) or small natural images (Krizhevsky, 2010). However, those data sets contain a large number of small images, where the number of images (number of data points) well exceeds the number of pixels per image (the dimension of a data point). For this experiment, we were interested if DBNs are able to learn the manifold of images without overfitting, when the dimension of a data point well exceeds the number of data points, as is commonly the case for biomedical data sets. A second question was if DBNs can learn meaningful patterns despite the limited amount of training data.
5.4.1 Data Sets and Preprocessing

We have evaluated the proposed method on a subset of the Alzheimer’s Disease Neuroimaging Initiative (ADNI) data set (Petersen et al., 2010), containing 300 T1-weighted (T1w) MRIs of AD and normal subjects. The images were provided skull-stripped and bias field corrected. We resampled all images to a resolution of $128 \times 128 \times 128$ voxels and a voxel size of $2.0 \times 2.0 \times 2.0$ mm. We then normalized their intensities to a common range, and rigidly registered them to a group-wise mean image prior to training and testing. We did not perform non-rigid registration for spatial normalization in order to evaluate the capabilities of the method without the added confound of complex registration parameters. The data set was divided into a training set and a test set such that each set contains 75 AD and 75 normal subjects.

5.4.2 Method

To learn the manifold of brain MRIs, we used a DBN with three sconvRBM layers and two dense RBM layers as described in Section 5.3.1. The parameters of the DBN were chosen to yield a continuous reduction of the dimensionality of the input images and are summarized in Table 5.1. For this experiment, we used circular convolutions instead of valid convolutions for the training of the sconvRBMs, which is a type of convolution that does not reduce the output size by the filter size. This simplifies the choice of parameters of the DBN, because the sizes of the hidden layers are independent of the filter sizes. For the first layer, we chose a stride size of $4 \times 4 \times 4$, which results in a strong reduction of the dimensionality in order to reduce the amount of memory required to train the second layer. Increasing the stride size decreases the overlap of adjacent sliding windows. To compensate, we also chose a larger filter size for the first layer than for all subsequent
Table 5.1: Parameters of the DBN used to learn the manifold of brain MRIs of AD and healthy subjects.

<table>
<thead>
<tr>
<th>Layer type</th>
<th>Kernel size</th>
<th>Stride size</th>
<th>#Filters/ #Hidden units</th>
<th>Output size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>128 × 128 × 128 × 1</td>
</tr>
<tr>
<td>sconvRBM1</td>
<td>20 × 20 × 20 × 1</td>
<td>4 × 4 × 4</td>
<td>32</td>
<td>32 × 32 × 32 × 32</td>
</tr>
<tr>
<td>sconvRBM1</td>
<td>14 × 14 × 14 × 32</td>
<td>2 × 2 × 2</td>
<td>32</td>
<td>16 × 16 × 16 × 32</td>
</tr>
<tr>
<td>sconvRBM1</td>
<td>10 × 10 × 10 × 32</td>
<td>2 × 2 × 2</td>
<td>64</td>
<td>8 × 8 × 8 × 64</td>
</tr>
<tr>
<td>RBM1</td>
<td>—</td>
<td>—</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>RBM2</td>
<td>—</td>
<td>—</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

layers. After the application of three sconvRBMs, the dimension of each image is reduced to 8 × 8 × 8 and small enough for RBMs. The training of the DBN took approximately 43 hours on two GeForce GTX 560 Ti graphics cards.

5.4.3 Results

Geometric Fit of the Manifold

The geometric fit of the learned manifold model was evaluated in terms of the generalizability to new images and the specificity to images from the training set. The generalizability was measured in terms of the average root mean squared error (RMSE) between images $I$ and their reconstructions $R$, normalized by the intensity range of the input images

$$RMSE = \sqrt{\frac{1}{N} \sum_p (I(p) - R(p))^2},$$

(5.2)

where $p$ denotes the coordinates of a particular voxel of an image and $N$ denotes the total number of voxels. The specificity was measured by calculating the average RMSE between images randomly generated from the manifold model using block Gibbs sampling and the most similar images from the training set. Figure 5.2 shows a comparison of the
reconstruction errors between the training and test sets, and the specificity at different layers of the DBN. The similarity of the reconstruction errors between the training and test images indicates that no overfitting is occurring. The average reconstruction error at the last layer is below 6%. Even though the very small reconstruction error is partially due to head MRIs having a large amount of homogeneous background, it demonstrates the ability of the learned manifold to capture most of the visual information with only two manifold parameters. The opposite slopes of the reconstruction errors and error of generated images indicates a trade-off between generalizability and specificity in the earlier phases of training. The low errors at the end of training (layer 5) indicates that the method is able to be both specific and generalizable.
Figure 5.3: Axial slices from generated volumes from the manifold. An increase of the first and second manifold dimension visually correlates with an increase in brain and ventricle size, respectively.

Visualization of the Learned Manifold

Figure 5.3 shows axial slices of 16 volumes sampled at the grid points of a 2D regular grid in manifold space. Volumes sampled along the first manifold dimension $M_1$ (from left to right) appear to increase in brain size, and the images sampled along the second manifold dimension $M_2$ (from bottom to top) appear to increase in ventricle size. Figure 5.4 shows an axial slice of each image of the training set plotted against its manifold coordinates. Consistent with images sampled from the manifold, an increase in ventricle size, which is indicative of brain atrophy (a hallmark of AD), visually correlates with an increase of the second manifold coordinate. The AD/normal status is indicated by the frame colour of each image. The vertical separation between AD and normals suggests that the second
Figure 5.4: Axial slices of volumes from the training set plotted against their manifold coordinates. The brains with larger ventricles, indicative of atrophy, are mostly at the top, which is also where most of the AD patients are.

manifold coordinate is potentially of practical use in differentiating between AD and normal.

Correlations with Clinical Parameters

To evaluate the potential of the manifold coordinates to reveal or predict clinically relevant information, we have calculated the Pearson correlation $r$ of demographic parameters (age, gender) and disease parameters (MMSE score, AD/normal status) with the manifold coordinates ($M_1$ and $M_2$). The results of the correlation tests are summarized in Table 5.2. Age, MMSE and AD/normal status show highly significant correlations with $M_2$ ($p$-values
Table 5.2: Pearson correlation $r$ of demographic and clinical parameters with manifold coordinates ($M_1$, $M_2$). The stronger correlation in each column is highlighted in bold. The level of statistical significance is indicated by the number of asterisks (* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$)

<table>
<thead>
<tr>
<th></th>
<th>Age</th>
<th>Gender</th>
<th>MMSE</th>
<th>AD/normal status</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>$-0.173^*$</td>
<td>0.449***</td>
<td>0.012</td>
<td>$-0.032$</td>
</tr>
<tr>
<td>$M_2$</td>
<td>0.447***</td>
<td>0.186*</td>
<td>$-0.402$***</td>
<td>0.413***</td>
</tr>
</tbody>
</table>

between $9.85 \times 10^{-9}$ and $3.53 \times 10^{-7}$, which makes intuitive sense because $M_2$ visually correlates with ventricle size. The first manifold coordinate correlates strongest with gender ($p$-value $= 8.24 \times 10^{-9}$), which also makes sense in terms of the general difference in size between male and female. The strength and significance of the correlations demonstrate the potential of deep learning of brain images for classification and prediction of disease status.

5.5 Variability of Morphology and Lesion Distribution

In this section, we present a new method for modelling the variability in brain morphology and lesion distribution in a large set of MRIs of MS patients. Similar to our previous work on modelling the manifold of brain MRIs of AD and healthy subjects, our method is built using a DBN (HINTON et al., 2006), a layered network whose parameters can be learned from training images. However, a difference to the previous approach is that the DBN is trained on deformation fields and lesions masks, instead of brain MRIs, in order to focus on the learning of the variability in brain morphology and lesion distribution, without the confounding influence of intensity variations. An advantage of DBNs over other manifold learning methods is that it does not require a prebuilt proximity graph, which is particularly beneficial for modelling lesions, because the sparseness and randomness of MS lesions make defining a suitable distance measure challenging and potentially biasing.
Instead, the DBN approach assumes that a particular lesion configuration is a sample from an unknown distribution of lesion configurations that can be parameterized by a relatively small set of lesion distribution parameters. We model morphological and lesion variability with two individual DBNs, then form a joint model by replacing the individual top network layers with a new layer that joins both DBNs, similarly to the work on the joint modelling of auditory and visual signals for speech recognition (Ngiam et al., 2011). Our results show that this model can automatically discover the classic patterns of MS pathology, as well as more subtle ones, and that the distribution parameters computed are found to have strong relationships to MS clinical scores.

5.5.1 Data Sets and Preprocessing

The proposed method was evaluated on a data set from an MS clinical trial of 474 secondary progressive MS patients. For each subject, the data set contains one T1w, T2-weighted (T2w), and proton density-weighted (PDw) MRI with a resolution of $256 \times 256 \times 50$ voxels and a voxel size of $0.937 \times 0.937 \times 3.000$ mm. The main preprocessing steps included rigid registration, brain extraction, intensity normalization, and background cropping.

5.5.2 Method

Our proposed model for pattern discovery is illustrated in Figure 5.5 and consists of three main components (a) a model that aims to find patterns of morphological changes in deformation fields (top pathway), (b) a model that aims to find patterns in the spatial distribution of lesions (bottom pathway), and (c) a joint model that aims to find concurrently deformation and lesion distribution patterns (combined pathways).
Figure 5.5: Proposed model for discovering patterns of variability in MS brain. Our model consists of three main components (a) a model that aims to find patterns of morphological changes in deformation fields (top pathway), (b) a model that aims to find patterns in the spatial distribution of lesions (bottom pathway), and (c) a joint model that aims to find concurring deformation and lesion distribution patterns (combined pathways).

Morphology Model

The morphology model is learned from a set of displacement fields that are calculated via non-rigid registration from a set of T1w brain MRIs $D \subset I$, $I = \{I_n \mid I_n : \Omega \rightarrow \mathbb{R}\}$, $\Omega \subset \mathbb{R}^3$ and the ICBM 152 nonlinear atlas template image (Fonov et al., 2011). First, all images of the training set are aligned to MNI space by a 9 degree-of-freedom registration using the FSL linear image registration tool (FLIRT) (Jenkinson et al., 2002). Then for each image $I_n \in D$, the displacement field $u_n$, $u : \Omega \rightarrow \mathbb{R}^3$, that describes the non-rigid transformation from template coordinates to image coordinates is calculated using the
FSL non-linear image registration tool (FNIRT) (Andersson et al., 2007), where the displacement field $u_n$ is represented by a $40 \times 48 \times 22$ grid of 3D displacement vectors. We assume that the displacement fields $u_n$ are samples from an unknown distribution $p(u \mid D_1, \ldots)$ that can be parameterized by far fewer parameters than the dimensionality of the fields themselves. In practice, the user typically selects the number of parameters to represent the data being explored. The task of finding patterns is to discover the underlying probability density function $p(u \mid D_1, \ldots)$, where the parameters $(D_1, \ldots)^T$ represent the patterns of variability of the displacement field distribution. This allows us to compare the morphology of two brains at a very high level in terms of the distribution parameters of their displacement fields $u_1$ and $u_2$ given by $E[D_1, \ldots \mid u_1]$ and $E[D_1, \ldots \mid u_2]$. Furthermore, we can visualize the modes of morphological variability of MS brain images, by sampling the space of displacement fields spanned by $(D_1, \ldots)^T$ by calculating the expectation $E[u \mid D_1, \ldots]$ for a range of values for $(D_1, \ldots)^T$.

The probability density function $p(u)$ is modelled by a DBN (Hinton et al., 2006). The first RBM layer receives the displacement fields of a training set as input and reduces the dimensionality of each field by discovering patterns of similarity that are common within groups of displacement fields. Each subsequent RBM receives the hidden unit activations of the previous RBM as input, thus learning successively more complex and abstract patterns from the training data. In particular, we use a DBN with three sconvRBMs and two dense RBMs as described in Section 5.3.1. Compared to other more commonly used convolution-based RBMs (Lee et al., 2009), an advantage of sconvRBMs is that inference is invertible, which allows the reconstruction of the visible units from the hidden unit activations. In our application, this would allow for the reconstruction of deformation fields from distribution parameters.
Table 5.3: Training parameters of the morphology DBN, lesion DBN, and joint DBN. The visible units of the joint DBN are composed of the concatenated hidden units of the first dense RBMs of the morphology and lesion DBNs.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Parameter</th>
<th>Morphology DBN</th>
<th>Lesion DBN</th>
<th>Joint DBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Image size</td>
<td>$40 \times 48 \times 22 \times 3$</td>
<td>$160 \times 192 \times 56 \times 1$</td>
<td>—</td>
</tr>
<tr>
<td>sconvRBM1</td>
<td>Stride size</td>
<td>$2 \times 2 \times 1$</td>
<td>$4 \times 4 \times 2$</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Filter size</td>
<td>$10 \times 10 \times 7$</td>
<td>$20 \times 20 \times 10$</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Filters</td>
<td>32</td>
<td>32</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Output size</td>
<td>$16 \times 20 \times 16 \times 32$</td>
<td>$36 \times 44 \times 24 \times 32$</td>
<td>—</td>
</tr>
<tr>
<td>sconvRBM2</td>
<td>Stride size</td>
<td>$2 \times 2 \times 2$</td>
<td>$2 \times 2 \times 2$</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Filter size</td>
<td>$10 \times 10 \times 10$</td>
<td>$14 \times 14 \times 10$</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Filters</td>
<td>64</td>
<td>64</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Output size</td>
<td>$4 \times 6 \times 4 \times 64$</td>
<td>$12 \times 16 \times 8 \times 64$</td>
<td>—</td>
</tr>
<tr>
<td>sconvRBM3</td>
<td>Stride size</td>
<td>$1 \times 1 \times 1$</td>
<td>$2 \times 2 \times 2$</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Filter size</td>
<td>$3 \times 5 \times 3$</td>
<td>$10 \times 14 \times 6$</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Filters</td>
<td>32</td>
<td>64</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Output size</td>
<td>$2 \times 2 \times 2 \times 32$</td>
<td>$2 \times 2 \times 2 \times 64$</td>
<td>—</td>
</tr>
<tr>
<td>RBM1</td>
<td>Visible units</td>
<td>256</td>
<td>512</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Hidden units</td>
<td>16</td>
<td>16</td>
<td>—</td>
</tr>
<tr>
<td>RBM2</td>
<td>Visible units</td>
<td>16</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Hidden units</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

The parameters of the morphology DBN are summarized in Table 5.3. In contrast to the DBN used to learn the manifold of brain MRIs of AD and healthy subjects, we use valid convolutions for the sconvRBMs, because they allow a much faster reduction of the dimensionality, especially for layers two and three, where the image sizes are relatively small compared to the filter sizes. This allows for a greater reduction of dimensionality within the sconvRBM layers, and consequently for a more gradual reduction of the number of hidden units during the transition of sconvRBMs to dense RBMs. To roughly compensate for the anisotropic voxel size of the input images and corresponding deformation fields, we chose an anisotropic filter and stride size of $10 \times 10 \times 7$ and $2 \times 2 \times 1$, respectively.
After three sconvRBMs, the size of a deformation field is reduced to $2 \times 2 \times 2 \times 32$ and small enough for RBMs.

Once the weights and bias terms of the DBN have been learned from training data, we can use the model for inference. Let $v_{d,1}, \ldots, v_{d,5}$, $h_{d,1}, \ldots, h_{d,5}$, and $\theta_{d,1}, \ldots, \theta_{d,5}$ denote the visible units, hidden units, and parameters, respectively, of each RBM of the morphology DBN. Then, for a given displacement field $u_n$, we can calculate the parameters $(D_1, \ldots)^T$ of $u_n \sim p(u \mid D_1, \ldots)$ with

$$
(D_1, \ldots)^T = E[D_1, \ldots \mid u_n] = E[h \mid v_{d,5}, \theta_{d,5}] 
$$

(5.3)

$$
v_{d,i+1} = E[h \mid v_{d,i}, \theta_{d,i}] 
$$

(5.4)

where $i \in [1, 4]$ and $v_{d,1} = u_n$. Inversely, the mean displacement field $\bar{u}$ given the distribution parameters can be calculated by

$$
\bar{u} = E[u \mid D_1, \ldots] = E[v \mid h_{d,1}, \theta_{d,1}] 
$$

(5.5)

$$
h_{d,i} = E[v \mid h_{d,i+1}, \theta_{d,i+1}] 
$$

(5.6)

where $i \in [1, 4]$ and $h_{d,5} = (D_1, \ldots)^T$.

**Lesion Model**

The input into our lesion model is a set of 3D binary lesion masks $l_n \in l$, which have been created from T2w and PDw MRIs by experts using a semi-automatic method. All lesion masks are spatially aligned to MNI space using the transformations as calculated for the corresponding T1w images. Analogous to the morphology model, we assume that lesion masks $l_n$ are samples from an unknown distribution $l_n \sim p(l \mid L_1, \ldots)$ that
can be parameterized by only relatively few parameters \((L_1, \ldots)^T\). The task of finding lesion patterns is to discover the underlying probability density function \(p(l \mid L_1, \ldots)\), where the parameters \((L_1, \ldots)^T\) represent patterns of variability of MS lesions. Similar to the morphology DBN, the lesion DBN consists of three sconvRBMs and two dense RBMs, whose parameters are summarized in Table 5.3. For the first layer, we chose an anisotropic stride size of \(4 \times 4 \times 2\), to roughly compensate for the anisotropic voxel size of the input images. A limiting factor for the choice of parameters is the amount of available GPU memory. Due to the much higher resolution of the lesion masks compared to the deformation fields, we increased the stride size compared to the morphology DBN in order to reduce the required amount of memory for storing the hidden units of the first layer and visible units of the second layer. Choosing a stride size of \(4 \times 4 \times 2\) reduces the size of one channel by a factor of roughly 32, which allows the use of 32 filters, while keeping the amount of memory required to train the second layer roughly equal compared to the first layer. To account for the sparse activation of MS lesion masks, we constrain the filter learning process to a pre-defined ROI, which was chosen to contain all white matter lesions from the training set. Similarly to the morphology model, for a trained lesion DBN and a given lesion mask \(l_n\), we can calculate the parameters \((L_1, \ldots)^T\) of \(l_n \sim p(l \mid L_1, \ldots)\) by

\[
(L_1, \ldots)^T = E[L_1, \ldots \mid l_n] = E[h \mid v_{l,5}, \theta_{l,5}]
\]

\[
v_{l,i+1} = E[h \mid v_{l,i}, \theta_{l,i}]
\]

\(107\)
where $i \in [1, 4]$ and $v_{l,1} = l_n$. Likewise, the mean lesion configuration $\bar{l}$ given the distribution parameters $(L_1, \ldots)^T$ can be calculated by

$$\bar{l} = E[l \mid L_1, \ldots] = E[v \mid h_{l,1}, \theta_{l,1}] \quad (5.9)$$

$$h_{l,i} = E[v \mid h_{l,i+1}, \theta_{l,i+1}] \quad (5.10)$$

where $v_{l,1}, \ldots, v_{l,5}$, $h_{l,1}, \ldots, h_{l,5}$, and $\theta_{l,1}, \ldots, \theta_{l,5}$ denote the visible units, hidden units, and parameters of each RBM of the lesion DBN, respectively, $i \in [1, 4]$, and $h_{l,5} = (L_1, \ldots)^T$.

Joint Model

To discover concurring patterns of morphology and lesion distribution, we combine the morphology DBN and the lesion DBN to form the joint DBN, which defines the joint distribution $p(u, l \mid J_1, \ldots)$. The joint DBN consists of two pathways (see Figure 5.5), each corresponding to the first 4 layers of the morphology and lesion DBNs, respectively, and a 5th RBM layer with 4 hidden units, which replaces the 5th layer of the individual DBNs and combines the hidden unit activations of the 4th layer RBMs. That is, the 5th RBM defines the joint probability $p(v_j, h_j \mid \theta_j)$, where the vector $v_j$ contains the concatenated hidden units of the 4th layer RBMs, $v_j = (h_{d,4}^T, h_{l,4}^T)^T$, and $h_j = (J_1, \ldots)^T$ are the modes of variability of morphological and lesion distribution changes.

5.5.3 Results

Visualization of Patterns of Variability

The invertibility of our model allows the reconstruction of images from the distribution parameters to visualize the discovered patterns of variability. Figure 5.6(a) shows axial
Chapter 5 Manifold Learning by Deep Learning

Figure 5.6: Slices from generated volumes from the (a) morphology, (b) lesion, and (c) joint model. The morphology model captures ventricular enlargement ($D_1$) and decrease in brain size ($D_2$) as the main modes of variation. For the lesion model, $L_1$ captures an increase in lesion load throughout the WM, while $L_2$ captures primarily periventricular lesion load variations. The parameters of the joint model capture combinations of the variability found in the individual models.

Slices from volumes generated from the 2-parameter morphology model $p(u \mid D_1, D_2)$. To generate these images, we calculated the mean displacement fields for varying values of $D_1$ and $D_2$ to span the range of variability represented by the training set and applied the inverse deformations to the ICBM 152 template image. The most apparent morphological variability captured by the morphology model is ventricular enlargement for $D_1$ and overall brain size for $D_2$. Figure 5.6(b) shows axial slices from the mean lesion configurations $E[l \mid L_1, L_2]$ for varying lesion distribution parameters. An increase of $L_2$ visually correlates with an increase of lesions specifically around the ventricles, whereas an increase of $L_1$ visually correlates with an increase of lesions in the entire white matter.

To visualize concurring patterns of morphology and lesion distribution, we sampled images from the joint model $p(u,l \mid J_1, \ldots, J_4)$ as shown in Figure 5.6(c). The images are
Table 5.4: Pearson correlations $r$ of clinical scores with distribution parameters of the morphology model ($D_1, D_2$), lesion model ($L_1, L_2$), joint model ($J_1, J_2, J_3, J_4$), normalized brain volume (nBV), and lesion load (LL). The level of statistical significance is indicated by the number of asterisks ($^* p < 0.05$, $^{**} p < 0.01$, $^{***} p < 0.001$).

<table>
<thead>
<tr>
<th></th>
<th>T25W</th>
<th>9-HPT</th>
<th>PASAT</th>
<th>MSFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>models</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D_1$</td>
<td>$-0.129^{**}$</td>
<td>$-0.215^{***}$</td>
<td>$-0.282^{***}$</td>
<td>$-0.315^{***}$</td>
</tr>
<tr>
<td>$D_2$</td>
<td>0.087</td>
<td>0.116*</td>
<td>0.089</td>
<td>0.139**</td>
</tr>
<tr>
<td>$L_1$</td>
<td>$-0.058$</td>
<td>$-0.231^{***}$</td>
<td>$-0.392^{***}$</td>
<td>$-0.367^{***}$</td>
</tr>
<tr>
<td>$L_2$</td>
<td>$-0.091$</td>
<td>$-0.354^{***}$</td>
<td>$-0.427^{***}$</td>
<td>$-0.464^{***}$</td>
</tr>
<tr>
<td>Joint</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>model</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$J_1$</td>
<td>0.107*</td>
<td>0.286***</td>
<td>0.336***</td>
<td>0.379***</td>
</tr>
<tr>
<td>$J_2$</td>
<td>$-0.038$</td>
<td>$-0.210^{***}$</td>
<td>$-0.227^{***}$</td>
<td>$-0.256^{***}$</td>
</tr>
<tr>
<td>$J_3$</td>
<td>$-0.118^{*}$</td>
<td>$-0.369^{***}$</td>
<td>$-0.453^{***}$</td>
<td>$-0.494^{***}$</td>
</tr>
<tr>
<td>$J_4$</td>
<td>$-0.049$</td>
<td>$-0.206^{***}$</td>
<td>$-0.383^{***}$</td>
<td>$-0.346^{***}$</td>
</tr>
<tr>
<td>Imaging</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>biomarkers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nBV</td>
<td>0.053</td>
<td>0.144**</td>
<td>0.247***</td>
<td>0.235***</td>
</tr>
<tr>
<td>LL</td>
<td>$-0.074$</td>
<td>$-0.286^{***}$</td>
<td>$-0.400^{***}$</td>
<td>$-0.406^{***}$</td>
</tr>
</tbody>
</table>

deformed template images with superimposed lesion masks. For each row, we varied only one distribution parameter and set the remaining parameters to their mean values. Of the four parameters, $J_3$ visually corresponds most closely to the “classical” progression of MS pathology, with an enlargement of the ventricles paired with an increased periventricular lesion load. The parameters $J_2$ and $J_4$ also reveal simultaneous morphological and lesion variations that are visible on the chosen axial slice. For $J_1$, a lesion pattern is not obvious unless the images are viewed sagittally, which reveals changes in lesion load in the pons.

**Correlations with Clinical Scores**

To evaluate the potential of the distribution parameters to reveal clinically relevant information, we have calculated the Pearson correlation $r$ of the clinical MS Functional Composite (MSFC) score (FISCHER et al., 1999) and its components (Timed 25-Foot Walk, T25W; 9-Hole Peg Test, 9-HPT; Paced Auditory Serial Addition Test, PASAT) with the distribution parameters and two established MS imaging biomarkers (normalized brain
volume, nBV, as calculated by SIENAX (Smith et al., 2002) and lesion load, LL). The results of the correlation tests are summarized in Table 5.4. In the individual models, all parameters correlate significantly with 9-HPT, PASAT, and MSFC, except for $D_2$, which did not show a statistically significant correlation with PASAT. The morphology parameter $D_1$ correlates more strongly with these scores than nBV, as does the lesion parameter $L_2$ than LL. For T25W, $D_1$ shows a modest but significant correlation while nBV does not. In the joint model, all parameters correlate significantly with 9-HPT, PASAT, and MSFC, with $J_3$ being particularly strong. The parameter $J_3$ shows stronger correlations than nBV or LL for all clinical scores, including a modest significant correlation to T25W, which is not shown by nBV nor LL. The significant correlation between $J_1$ and T25W is particularly interesting because the lesion changes modelled by $J_1$ occur in the pons, which is known to be of clinical significance for mobility. Our model captures the largest variability in the data set and inter-subject variability is a possible confound. However, the stronger correlations compared to nBV and LL suggest that disease variability is more dominant.
Progression of a “Mean” SPMS Brain along a Range of MSFC Scores

Another benefit of our model is the ability to visualize the progression of a “mean” secondary progressive MS (SPMS) brain along a range of MSFC scores. To demonstrate, we trained four independent linear models to predict the distribution parameters \( J_1, \ldots, J_4 \) given the MSFC \( J_i = a_i + b_i \text{MSFC} \). Figure 5.7 shows the axial (top row) and mid-sagittal (bottom row) slices of generated images representing the range of MSFC scores from 1.5 to −4.5. Consistent with previous results, a decrease in MSFC visually correlates with an increase in the size of the ventricles, an increase in periventricular lesions, and an increase in lesions in the pons region.

5.6 Summary

We have introduced a new method for modelling the variability in brain morphology and lesion distribution of a large set of MRIs of AD and MS patients. We have proposed two models, both of which are based on DBNs. In our first approach, variability of brain morphology is modelled as a manifold of brain MRIs. In our second approach, we train three DBNs: one for morphology, one for lesion distribution, and one that jointly models both. The morphology model is learned from deformation fields, which allows the algorithm to focus on structural changes without the confound of contrast variations between images. We have demonstrated that such a model, which requires no built-in priors on image similarity, can automatically discover patterns of variability that can be parameterized in a low-dimensional space and are clinically relevant. In addition, our model can generate sample images from model parameters for visualization.
In this thesis, we have addressed two challenges for measuring the state and progression of multiple sclerosis (MS). In Chapter 4, we presented a fully automatic segmentation method of MS lesions, which enables the accurate computation of lesion-based imaging biomarkers such as lesion load and lesion count. In Chapter 5, we presented a novel method for the automatic discovery of patterns of variability in MS subjects that show improved correlations to clinical scores over traditional volumetric biomarkers. Both methods are based on deep learning, a class of algorithms that can learn a feature hierarchy from a set of images. A major challenge of deep learning methods is the time and memory required to process large 3D volumes. To address this challenge, we have developed a novel training algorithm for convolutional deep learning methods that facilitates the training of full resolution 3D volumes and presented our algorithm with a comprehensive comparison with alternative training methods in Chapter 3. Improving the training speed is a rapidly advancing topic, but there are still very few publications that use images of such high dimensionality, which suggests that computational speed is still a bottleneck in general. Although the developed methods were developed and evaluated in the context of
MS, the methods are very general and can potentially be applied to various segmentation and manifold learning problems.

### 6.1 Contributions

In the course of developing deep learning-based methods for MS lesion segmentation and pattern discovery, we have made the following main contributions:

1. We have developed a novel training algorithm for convolutional deep belief networks and convolutional neural networks that performs training in the frequency domain. The speed-ups gained by our method compared to state-of-the art spatial domain implementations and the reduced memory requirements compared to other frequency domain methods enable the application of deep learning to high-resolution 3D medical images.

2. We have developed a neural network architecture that jointly learns features at different scales that are tuned to segmenting MS lesions and performs the segmentation based on the automatically learned features. The joint learning of feature extractor and classifier facilitates the learning of features that are robust to the large variability of MS lesions and varying contrasts produced by different scanners.

3. We have developed a novel objective function for training neural networks that is suitable for the classification of vastly unbalanced classes, such as the segmentation of MS lesions, which typically comprise less than one percent of the image.

4. This is the first work to demonstrate that deep learning can be applied to manifold learning of brain MRIs.
5. We have developed a framework for modelling changes in brain morphology and lesion distribution with only a few parameters, which also show improved correlation with clinical scores compared to established volumetric imaging biomarkers.

6.2 Future Work

6.2.1 New Applications of Deep Learning

Motivated by the success of deep learning for the two applications explored in this theses, a possible direction for further research is to investigate new applications of deep learning for medical image analysis. The success of deep learning methods mostly stems from their ability to learn feature hierarchies directly from the data. This allows deep learning methods to adapt to the challenges presented by the data, such as imaging artifacts and contrast variations, without explicitly having to model them. However, deep learning methods can take a lot of time to train and require large amounts of training data in order to learn a feature representation that generalizes well to new data samples. To overcome those challenges, first applications of deep learning for medical image analysis were limited to problems that can be cast as a patch-wise classification problem, such as the segmentation of cell membranes (Cireşan et al., 2012) and the detection of cell carcinoma (Cruz-Roa et al., 2013). In this fashion, every patch of an image is considered a training sample, which greatly increases the amount of labelled training cases. In addition, training on relatively small 2D patches made training feasible. Similarly, fully convolutional approach for image segmentation leverage every pixel of an image as a training sample, while being more computationally efficient than patch-based approaches. However, the number of voxels within a layer decreases with the depth of the network,
with the result that deeper layers are trained with fewer training samples, which increases the risk of overfitting.

A possible direction for future research is to investigate the application of the proposed deep learning segmentation framework to other segmentation tasks. Preliminary results on the segmentation of the corpus callosum and the grey matter of the spinal cord show great promise. Another potential application could be the detection of landmark points. Detecting landmarks is similar to segmentation in that every voxel of an image can be leveraged as a training sample. However, the number of positive samples per image is typically much smaller than for segmentation, which might require the development of alternative training approaches.

A different direction for future research is to investigate the potential of deep learning methods to be incorporated into existing model learning frameworks. Originally, active shape (Cootes et al., 1995) and active appearance (Cootes et al., 2001) models employ principal component analysis in order to reduce the dimensionality of the input feature space. Similarly, our proposed manifold learning framework can be used for dimensionality reduction, which might yield to the learning of more biologically plausible models, due to their ability to find highly nonlinear patterns of variability in the data.

6.2.2 Improving the Performance on Small Data Sets using Data Augmentation

A promising approach for improving the performance on small data sets is data augmentation. Data augmentation allows the artificial increase of the training data set size by generating new training samples from existing ones. Data augmentation has been successfully used to deal with the problem of small medical data sets. For example, Ronneberger et al. (2015) used automatically generated deformations to artificially increase the data set size for segmenting neuronal structures in electron microscopic stacks, but also
simpler data augmentation techniques such as adding rotations, translations and contrast variations can help to avoid overfitting. However, hand-engineered data augmentation approaches depend on how well the variability in the data is understood. Alternatively, automatically learned unsupervised deep generative models such as deep belief networks can potentially be used to learn the variability of the training data in order to generate new samples that are biologically plausible.

6.2.3 Rotation-invariant Neural Networks

Although neural networks are able to learn features that are invariant to the variability in the data, doing so often requires the learning of multiple variants of the same feature in order to capture the range of variability. In order to decrease the number of weights of a neural network, and therefore to reduce the training time and risk of overfitting, different neural network architectures have been developed with build-in invariance or equivariance properties. For example, previously used sigmoid activation functions are sensitive to the intensity range of the input, which requires the relearning of feature detectors for images with varying contrast. To overcome, rectified linear units have been proposed, which are equivariant to intensity variations. In order to allow the learning of translation-invariant features, convolutional neural networks have been proposed, which greatly reduce the number of weights required to capture reoccurring patterns at different locations in the image compared to dense neural networks. However, features learned by a convolutional neural network are not invariant to rotations, which leads to the relearning of the same class of features at different poses, and consequently increases the number of filters required to sample the pose space. This problem is especially pronounced in 3D. While sampling one feature under 2D rotations in 30 degree steps leads to 12 samples, it leads to 1728 samples for full 3D rotations, as three angles are required to determine a
3D pose. In order to reduce the number of filters required to capture structures in 3D, it might therefore be necessary to develop neural network architectures that are inherently invariant or equivariant to rotations, which would allow for the learning of a diverse set of features that capture fine details in 3D with much fewer filters than currently employed convolutional neural networks.

6.2.4 Increasing the Expressive Power of Single Neurons

Current neural networks can only calculate the weighted sum of its inputs followed by the application of a nonlinear function. Although it has been shown that this model is able to approximate arbitrary functions (Cybenko, 1989), doing so might require a large number of units and layers, which makes them more difficult to train and requires more training data in order to reduce the risk of overfitting. An alternative for increasing the expressive power of neural networks without increasing their complexity is to design more computationally powerful units. In their review of the computational properties of dendrites, London and Häusser (2005) noted that some neurons were found that are able to calculate the multiplication of its inputs. For example, the looming sensitive neurons in the locust’s visual system are able to detect approaching objects on a collision course by calculating the multiplication of angular size and speed of approaching object (London and Häusser, 2005). The multiplication is possibly encoded as the sum of logarithmic inputs followed by exponentiation. Similarly, Godfrey and Gashler (2016) have proposed a type of neuron for artificial neural networks that is able to learn to calculate the logarithm or exponential of its inputs, which allows for the learning of networks that can calculate arbitrary polynomials with significantly fewer parameters than traditional rectified linear or sigmoid neural networks. Alternatively, Livni et al. (2013) have proposed a network architecture that can learn arbitrary polynomials using a combination of units that calculate
weighted sums of its inputs and units that calculate the product of two input numbers. However, these network architectures need to be investigated further in order to show if the theoretical advantages translate into improvements on practical problems.
Bibliography


