DEEP LEARNING FOR MANIFOLD LEARNING AND SEGMENTATION OF BRAIN MRIS

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)

November 2015

Abstract

Brosch, Tom:
Deep Learning for Manifold Learning and Segmentation of Brain MRIs
PhD thesis, The University of British Columbia, 2015

Contents

List of Figures

1 Introduction

Introduction of deep learning and motivation. What is deep learning? Why is everyone so hyped about it? What is the vision/promise? Thinking in feature hierarchies. Neuroplasticity suggestions that this can be learned. Inspiration for different models like neural networks. Old field but met with problems. Renaissance with faster hardware, more data, layerwise pre-training. Latest development is fast. Fill this with examples of different domains and how deep learning helped to revolutionized object recognition, object detection, semantic segmentation, speech recognition, natural language processing.

Goal of the thesis. What am I up to? If it was so successful in so many different domain, can deep learning help to solve problems in the medical domain and where can it have a big impact? In this thesis, we focus on two possible applications of deep learning, unsupervised learning of features that are descriptive of images as a whole. This is directly related to manifold learning and learning the variability in brain MRIs and the relationship to diseases. The other part is image segmentation with the focus on MS lesion segmentation.

Give an outlook. First introduce some deep learning methods. Then talk about special challenges when it comes to 3D medical images. A big part is training. Training in the frequency domain explained plus a discussion of related work. A general introduction

is followed by the applications of deep learning. This will be a chapter on its own. Start with learning variability in images or global image descriptors and then segmentation.

1.1 Multiple Sclerosis

Give some introduction to MS and why it is important and what the challenges are and how that links to what I'm doing in this thesis.

2 Background

Background chapter on deep learning methods. Start with general introduction. I want to talk about some deep learning methods. Main goal is learning a feature hierarchy. Models are often biologically inspired. Neuroplasticity strong motivator for learning. Start with either unsupervised or supervised models. Explain what it means and why it is important. Unsupervised no need for labels, which can be difficult to obtain. Supervised tuned to task. Can perform better when enough labels are available. Requires careful regularization or lots of data because more prone to overfitting.

2.1 Learning from Labeled Data

2.1.1 Neural Networks

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

$$\mathbf{z}_l = \mathbf{W}_l \mathbf{x}_{l-1} + \mathbf{b}_l, \tag{2.1}$$

$$\mathbf{x}_l = f_l(\mathbf{z}_l), \tag{2.2}$$

where $\mathbf{x}_0$ denotes the input of the neural network, $\mathbf{x}_L$ denotes 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

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)}, \tag{2.3}$$

where $\mathbf{z}$ denotes an n-dimensional output vector.

Given a training set $\mathcal{D} = \{(\mathbf{x}_0^{(i)}, \mathbf{y}^{(i)}) \mid i \in [1, N]\}$, a neural network is trained by minimizing the error between the predicted outputs $\mathbf{x}_L^{(i)}$ and the given labels $\mathbf{y}^{(i)}$

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} E\left(\mathbf{x}_L^{(i)}, \mathbf{y}^{(i)}\right), \tag{2.4}$$

where $\boldsymbol{\theta}$ denotes the trainable parameters of the neural network. Typical choices for the error function are the sum of squared differences (SSD), and 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

$$\delta_L = \nabla_{\mathbf{x}_L} E \cdot f'_L(\mathbf{z}_L), \tag{2.5}$$

$$\delta_l = (\mathbf{W}_{l+1}^{\mathrm{T}} \delta_{l+1}) \cdot f'_l(\mathbf{z}_l) \qquad \text{for } l < L, \tag{2.6}$$

$$\nabla_{\mathbf{W}_l} E = \delta_l \mathbf{x}_{l-1}^{\mathrm{T}}, \tag{2.7}$$

$$\nabla_{\mathbf{b}_l} E = \delta_l, \tag{2.8}$$

where $\nabla_{\mathbf{x}_L} E$ 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 2.1), simple cells are represented by convolutional layers, which exhibit a similar mechanism of local connectivity and weight sharing. Complex cells combine the activation of simple cells to add robustness to small translations. These cells are represented in the form of pooling layers similar to the pooling layers found in convDBNs. 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 lattice, where the first three dimensions correspond to the dimensions of

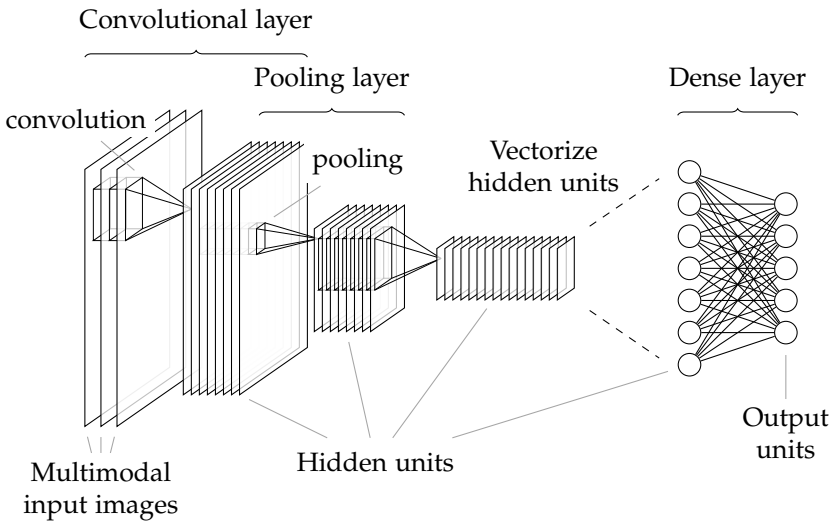


Figure 2.1: Convolutional neural network with two convolutional layers, one pooling layer and one dense layer. The activations of the last layer is the output of the network.

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

$$x_j^{(l)} = f\left(\sum_{i=1}^{C} \tilde{w}_{ij}^{(l)} * x_i^{(l-1)} + b_j^{(l)}\right), \tag{2.9}$$

where l is the index of a convolutional layer, $x_j^{(l)}$ denotes the jth channel of the output volume, $w_{ij}^{(l)}$ is a 3D filter kernel connecting the ith channel of the input volume to the jth channel of the output volume, and $b_j^{(l)}$ denotes the bias term of the jth output channel. 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).

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 (Duchi et al., 2011), AdaDelta (Zeiler, 2012), RMSprop (Dauphin et al., 2015), and Adam (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.

2.1.3 Training and pre-training

Problem with learning rate and stuff. Can be pre-trained.

2.2 Learning from Unlabeled Data

One of the most important applications of deep learning is to learn 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 (RBMs) (Freund and Haussler, 1992; **?**), which are the building blocks of deep belief networks (DBNs) (Hinton et al., 2006), followed by a short introduction to alternative feature extractors such as stacked denoising autoencoders (SDAEs) (Vincent et al., 2010).

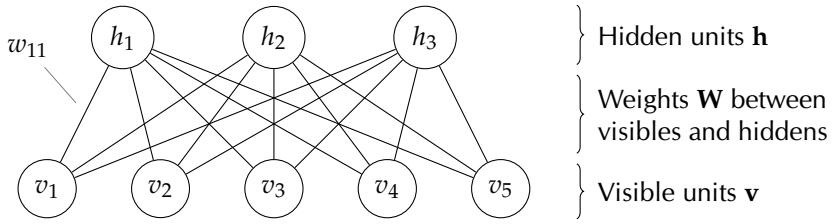


Figure 2.2: Graph representation of an RBM with 3 hidden and 5 visible units. An RBM models the joint probability of visible and hidden units. Edges between vertices denote conditional dependence between the corresponding random variables.

2.2.1 From Restricted Boltzmann Machines to Deep Belief Networks

An RBM is a probabilistic graphical models defined by a bipartite graph as shown in Fig. Figure 2.2. The units of the RBM are divided into two layers, one of visible units $\mathbf{v}$ and the other of hidden units $\mathbf{h}$. 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

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

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

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

$$= \mathbf{v}^{\mathrm{T}} \mathbf{W} \mathbf{h} + \mathbf{b}^{\mathrm{T}} \mathbf{v} + \mathbf{c}^{\mathrm{T}} \mathbf{h} \tag{2.12}$$

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

Inference The hidden units represent patterns of similarity that can be observed in groups of images. Once an RBM is trained, the features of a previously unseen 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

$$p(h_j = 1 \mid \mathbf{v}, \boldsymbol{\theta}) = \text{sigm}(\mathbf{w}_{\cdot,j}^{\mathrm{T}}\mathbf{v} + c_j), \tag{2.13}$$

where $\mathbf{w}_{\cdot,j}$ denotes the jth column vector of $\mathbf{W}$, 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 $p(v_i = 1 \mid \mathbf{h}, \boldsymbol{\theta})$ can be calculated by

$$p(v_i = 1 \mid \mathbf{h}, \boldsymbol{\theta}) = \text{sigm}(\mathbf{w}_{i,\cdot}^{\mathrm{T}}\mathbf{h} + b_i), \tag{2.14}$$

where $\mathbf{w}_{i,\cdot}$ denotes the ith row vector of $\mathbf{W}$. 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, $\mathcal{D} = \{\mathbf{v}_n \mid n \in [1, N]\}$, which is called maximum likelihood

estimation (MLE). The gradient of the log likelihood function with respect to the weights, $\mathbf{W}$, is given by

$$\nabla_{\mathbf{W}} \log p(\mathcal{D} \mid \boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}[\mathbf{v}\mathbf{h}^{\mathrm{T}} \mid \mathbf{v}_n, \boldsymbol{\theta}] - \mathbb{E}[\mathbf{v}\mathbf{h}^{\mathrm{T}} \mid \boldsymbol{\theta}]. \tag{2.15}$$

The first expectation can be estimated using a mean field approximation:

$$\mathbb{E}[\mathbf{v}\mathbf{h}^{\mathrm{T}} \mid \mathbf{v}_n, \boldsymbol{\theta}] \approx \mathbb{E}[\mathbf{v} \mid \mathbf{v}_n, \boldsymbol{\theta}]\mathbb{E}[\mathbf{h}^{\mathrm{T}} \mid \mathbf{v}_n, \boldsymbol{\theta}], \tag{2.16}$$

$$= \mathbf{v}_n \mathbb{E}[\mathbf{h}^{\mathrm{T}} \mid \mathbf{v}_n, \boldsymbol{\theta}]. \tag{2.17}$$

The second expectation is typically estimated using a Monte Carlo approximation

$$\mathbb{E}[\mathbf{v}\mathbf{h}^{\mathrm{T}} \mid \boldsymbol{\theta}] \approx \frac{1}{S} \sum_{s=1}^{S} \mathbf{v}_s \mathbf{h}_s^{\mathrm{T}}, \tag{2.18}$$

where S is the number of generated samples, and $\mathbf{v}_s$ and $\mathbf{h}_s$ are samples drawn from $p(\mathbf{v} \mid \boldsymbol{\theta})$ and $p(\mathbf{h} \mid \boldsymbol{\theta})$, 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

$$h_j = \mathbb{I}(y_j < p(h_j = 1 \mid \mathbf{v}, \boldsymbol{\theta})) \qquad \text{with } y_j \sim \mathrm{U}(0, 1) \tag{2.19}$$

$$v_i = \mathbb{I}(x_i < p(v_i \mid \mathbf{h}, \boldsymbol{\theta})) \qquad \text{with } x_i \sim \mathrm{U}(0, 1) \tag{2.20}$$

where $z \sim \mathrm{U}(0, 1)$ denotes a sample drawn from the uniform distribution in the interval $[0, 1]$ and $\mathbb{I}$ 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 $p(\mathbf{v}\mathbf{h} \mid \boldsymbol{\theta})$.

If the Gibbs sampler is initialized at a data point from the training set and only 1 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, the dataset 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 dataset. Additional tricks to monitor and speed up the training of an RBM can be found in Hinton's RBM training guide (Hinton, 2010).

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's activations computed from the previous RBM. The stacking of RBMs can be repeated to initialize DBNs of any depth.

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

$$-E(\mathbf{v}, \mathbf{h} \mid \boldsymbol{\theta}) = \sum_{i,j} \frac{v_i}{\sigma_i} w_{ij} h_j + \sum_i \frac{(v_i - b_i)^2}{2\sigma_i^2} + \sum_j c_j h_j, \tag{2.21}$$

where the mean of the ith visible unit is encoded in the bias term b_i, and its standard deviation is given by σ_i. Although approaches have been proposed to learn the standard deviation (Cho et al., 2011), the training data is often simply standardized to have zero mean and unit variance, which yields to the following simplification for the inference of the visible and hidden units:

$$\mathbb{E}[h_j \mid \mathbf{v}, \boldsymbol{\theta}] = \text{sigm}(\mathbf{w}_{\cdot,j}^{\mathrm{T}}\mathbf{v} + c_j), \tag{2.22}$$

$$\mathbb{E}[v_i \mid \mathbf{h}, \boldsymbol{\theta}] = \mathbf{w}_{i,\cdot}^{\mathrm{T}}\mathbf{h} + b_i. \tag{2.23}$$

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 (NReLU) (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:

$$\mathbb{E}[h_j \mid \mathbf{v}, \boldsymbol{\theta}] = \sum_{i=1}^{\infty} \text{sigm}(\mathbf{w}_{\cdot,j}^{\mathrm{T}}\mathbf{v} + c_j - i + 0.5), \tag{2.24}$$

$$\approx \log(1 + \exp(\mathbf{w}_{\cdot,j}^{\mathrm{T}}\mathbf{v} + c_j)). \tag{2.25}$$

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

$$h_j \sim \max(0, \mu_j + \mathcal{N}(0, \text{sigm}(\mu_j))), \tag{2.26}$$

$$\mu_j = \mathbf{w}_{\cdot,j}^{\mathrm{T}} \mathbf{v} + c_j, \tag{2.27}$$

where $\mathcal{N}(0, \sigma^2)$ denotes Gaussian noise.

2.2.2 Convolutional Models

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, have to 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 convolutional deep belief network (convDBN) (Lee et al., 2009, 2011). In a convDBN, the units of one layer are only connected to the units of a sub-region of the previous layer, where each unit shares 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.

We assume the input images to be square 2D images, but the model generalizes with little modification to non-square images of higher dimensions.

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.

Symbol	Description
$\mathbf{v}^{(i)}$	ith input channel
$\mathbf{h}^{(j)}$	jth output channel or feature map
$\mathbf{w}^{(ij)}$	weights of filter kernels connecting visible units $\mathbf{v}^{(i)}$ to hidden units $\mathbf{h}^{(j)}$
b_i	bias terms of visible units
c_j	bias terms of hidden units
N_c	number of channels of the visible units
N_v^2	number of visible units per channel
N_k	number of filters and feature maps
N_h^2	number of hidden units per feature map
$\bullet$	element-wise product followed by summation
$*$	valid convolution
$\circledast$	full convolution
$\tilde{\mathbf{w}}^{(ij)}$	horizontally and vertically flipped version of $\mathbf{w}^{(ij)}$

greedy layerwise fashion by treating it as a convolutional restricted Boltzmann machine (convRBM). The energy of a convRBM is defined as

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=1}^{N_\mathrm{c}} \sum_{j=1}^{N_\mathrm{k}} \mathbf{h}^{(j)} \bullet (\tilde{\mathbf{w}}^{(ij)} * \mathbf{v}^{(i)}) - \sum_{i=1}^{N_\mathrm{c}} b_i \sum_{x,y=1}^{N_\mathrm{v}} v_{xy}^{(i)} - \sum_{j=1}^{N_\mathrm{k}} c_j \sum_{x,y=1}^{N_\mathrm{h}} h_{xy}^{(j)}. \tag{2.28}$$

The key terms and notation are defined in Table Table 2.1. At the first layer, the number of channels N_c is equal to the number of input modalities. For subsequent layers, N_c is equal to the number of filters of the previous layer.

The posterior distributions $p(\mathbf{h} \mid \mathbf{v})$ and $p(\mathbf{v} \mid \mathbf{h})$ can be derived from the energy equation and are given by

$$p(h_{xy}^{(j)} = 1 \mid \mathbf{v}) = \text{sigm}\Big(\sum_{i=0}^{N_c-1} (\tilde{\mathbf{w}}^{(ij)} * \mathbf{v}^{(i)})_{xy} + c_j\Big), \tag{2.29}$$

$$p(v_{xy}^{(i)} = 1 \mid \mathbf{h}) = \text{sigm}\Big(\sum_{j=0}^{N_k-1} (\mathbf{w}^{(ij)} \circledast \mathbf{h}^{(j)})_{xy} + b_i\Big). \tag{2.30}$$

To train a convRBM on a set of images $\mathcal{D} = \{\mathbf{v}_n \mid n \in [1, N]\}$, 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

$$\Delta\mathbf{w}^{(ij)} \approx \frac{1}{N}(\mathbf{v}_n^{(i)} * \tilde{\mathbf{h}}_n^{(j)} - \mathbf{v}_n'^{(i)} * \tilde{\mathbf{h}}_n'^{(j)}), \tag{2.31}$$

where $\mathbf{h}_n^{(j)}$ and $\mathbf{h}_n'^{(j)}$ are samples drawn from $p(\mathbf{h}^{(j)} \mid \mathbf{v}_n)$ and $p(\mathbf{h}^{(j)} \mid \mathbf{v}_n')$, and $\mathbf{v}_n'^{(i)} = \mathbb{E}[\mathbf{v}^{(i)} \mid \mathbf{h}_n]$.

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.

2.2.3 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. This reduces the

number of hidden units per channel to $N_\mathrm{h} = N_\mathrm{v}/s$, hence significantly reducing training times and memory required for storing the hidden units during training. The energy of a strided convolutional RBM (sconvRBM) is defined as

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=0}^{N_\mathrm{c}-1} \sum_{j=0}^{N_\mathrm{k}-1} \sum_{x,y=0}^{N_\mathrm{h}-1} \sum_{u,v=-\lfloor N_\mathrm{w}/2 \rfloor}^{\lfloor (N_\mathrm{w}-1)/2 \rfloor} h_{xy}^{j} w_{uv}^{ij} v_{sx+u,sy+v}^{i} - \sum_{i=0}^{N_\mathrm{c}-1} b_i \sum_{x,y=0}^{N_\mathrm{v}-1} v_{xy}^{i} - \sum_{j=0}^{N_\mathrm{k}-1} c_j \sum_{x,y=0}^{N_\mathrm{h}-1} h_{xy}^{j} \tag{2.32}$$

2.2.4 Alternative Feature Learning Approaches

Stacked auto encoders and denoising autoencoders without math. Independent component analysis and independent subspace analysis. Need to read about this.

3 Training on Medical Images: Training in the Frequency Domain

Deep learning 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), 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. RainaEt al. (2009) demonstrated that by using graphics cards, training of restricted Boltzmann machines (RBMs) 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 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 to reduce the number of trainable weights. They showed that a convDBN can be used to classify images with

a resolution up to 200×200 pixels. To speed up the training of convolutional neural networks (CNNs) 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 256×256 pixels, achieving state-of-the-art performance on the ILSVRC-2010 and ILSVRC-2012 competitions (Krizhevsky et al., 2012). An alternative approach 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.

In this paper, we detail our training algorithm and GPU implementation in full, with a much more thorough analysis of the running time on high-resolution 2D images (512×512) and 3D volumes ($128 \times 128 \times 128$), showing speed-ups of up to 8-fold and 200-fold, respectively. Our 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. We show that our method can be efficiently implemented on multiple graphics cards, further improving the runtime performance over other GPU-accelerated training methods. In addition, we formalize the expression of the strided convolutional

DBN (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.

3.1 Algorithm

3.1.1 Training in the Spatial Domain

To train an sconvRBM on a set of images, 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

$$\Delta \mathbf{W}^{ij} \approx \frac{1}{N}(\mathbf{V}_n^i * \tilde{\mathbf{h}}_n^j - \mathbf{V}_n'^i * \tilde{\mathbf{h}}_n'^j) \tag{3.1}$$

where $\mathbf{V}_n, n \in [0, N-1]$ are reindexed images from the training set, $\mathbf{h}_n^j$ and $\mathbf{h}_n'^j$ are samples drawn from $p(\mathbf{h}^j \mid \mathbf{V}_n)$ and $p(\mathbf{h}^j \mid \mathbf{V}_n')$, and $\mathbf{V}_n'^i = \mathbb{E}[\mathbf{V}^i \mid \mathbf{h}_n]$. To apply the model to real-valued data like certain types of images, the visible units can be modeled as Gaussian units. When the visible units are mean–centered and standardized to unit variance, the expectation of the visible units is given by

$$\mathbb{E}[\mathbf{V}^i \mid \mathbf{h}] = \textstyle\sum_j \mathbf{W}^{ij} * \mathbf{h}^j + b_i \tag{3.2}$$

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 as the hidden units, which have

been shown to improve the learning performance of RBMs (**?**). The hidden units can be sampled with

$$\mathbf{h}^j \sim \max(0, \mu^j + \mathcal{N}(0, \mathrm{sigm}(\mu^j))) \tag{3.3}$$

$$\mu^j = \textstyle\sum_i \tilde{\mathbf{W}}^{ij} * \mathbf{V}^i + c_j \tag{3.4}$$

where $\mathcal{N}(0, \sigma^2)$ denotes Gaussian noise. The learning algorithm in the spatial domain is summarized in Figure 3.1a.

3.1.2 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 $5 \times N_\mathrm{C} \times N_\mathrm{k}$ 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. 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. 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 $\tilde{w}(a) = w(-a)$, which can be expressed by element-wise calculation of the complex conjugate; this follows directly from

input : Images $\mathcal{D} = \{\mathbf{V}_n \mid n \in [0, N-1]\}$
output: Weights gradient $\Delta\mathbf{W}$

$\Delta\mathbf{W} = 0$
foreach image $\mathbf{V} \in \mathcal{D}$ **do**
  $\mathbf{V}' = 0$
  for $j = 0$ **to** $N_\mathrm{k} - 1$ **do**
    $\mu^j = \sum_i \tilde{\mathbf{W}}^{ij} * \mathbf{V}^i + c_j$
    $h^j \sim \max(0, \mu^j + \mathcal{N}(0, \mathrm{sigm}(\mu^j)))$
    for $i = 0$ **to** $N_\mathrm{C} - 1$ **do**
      $\Delta\mathbf{W}^{ij} = \Delta\mathbf{W}^{ij} + \tilde{\mathbf{h}}^j * \mathbf{V}^i$
      $\mathbf{V}'^i = \mathbf{V}'^i + \mathbf{W}^{ij} * \mathbf{h}^j$
  $\forall i\colon \mathbf{V}'^i = \mathbf{V}'^i + b_i$
  for $j = 0$ **to** $N_\mathrm{k} - 1$ **do**
    $\mu'^j = \sum_i \tilde{\mathbf{W}}^{ij} * \mathbf{V}'^i + c_j$
    $\mathbf{h}'^j \sim \max(0, \mu'^j + \mathcal{N}(0, \mathrm{sigm}(\mu^j)))$
    for $i = 0$ **to** $N_\mathrm{C} - 1$ **do**
      $\Delta\mathbf{W}^{ij} = \Delta\mathbf{W}^{ij} - \tilde{\mathbf{h}}'^j * \mathbf{V}'^i$

(a) Training in the spatial domain

input : Images $\hat{\mathcal{D}} = \{\hat{\mathbf{V}}_n \mid n \in [0, N-1]\}$
output: Weights gradient $\Delta\hat{\mathbf{W}}$

$\Delta\hat{\mathbf{W}} = 0$
foreach image $\hat{\mathbf{V}} \in \hat{\mathcal{D}}$ **do**
  $\hat{\mathbf{V}}' = 0$
  for $j = 0$ **to** $N_\mathrm{k} - 1$ **do**
    $\hat{\mu}^j = \sum_i \overline{\hat{\mathbf{W}}^{ij}} \cdot \hat{\mathbf{V}}^i$
    $\mu^j = \mathrm{ifft}(\hat{\mu}^j) + c_j$
    $\mathbf{h}^j \sim \max(0, \mu^j + \mathcal{N}(0, \mathrm{sigm}(\mu^j)))$
    $\hat{\mathbf{h}}^j = \mathrm{fft}(\mathbf{h}^j)$
    for $i = 0$ **to** $N_\mathrm{C} - 1$ **do**
      $\Delta\hat{\mathbf{W}}^{ij} = \Delta\hat{\mathbf{W}}^{ij} + \overline{\hat{\mathbf{h}}^j} \cdot \hat{\mathbf{V}}^i$
      $\hat{\mathbf{V}}'^i = \hat{\mathbf{V}}'^i + \hat{\mathbf{W}}^{ij} \cdot \hat{\mathbf{h}}^j$
  $\forall i\colon \hat{V}'^i_{0,0} = \hat{V}'^i_{0,0} + N_\mathrm{V}^2 b_i$
  for $j = 0$ **to** $N_\mathrm{k} - 1$ **do**
    $\hat{\mu}'^j = \sum_i \overline{\hat{\mathbf{W}}^{ij}} \cdot \hat{\mathbf{V}}'^i$
    $\mu'^j = \mathrm{ifft}(\hat{\mu}'^j) + c_j$
    $\mathbf{h}'^j \sim \max(0, \mu'^j + \mathcal{N}(0, \mathrm{sigm}(\mu^j)))$
    $\hat{\mathbf{h}}'^j = \mathrm{fft}(\mathbf{h}'^j)$
    for $i = 0$ **to** $N_\mathrm{C} - 1$ **do**
      $\Delta\hat{\mathbf{W}}^{ij} = \Delta\hat{\mathbf{W}}^{ij} - \overline{\hat{\mathbf{h}}'^j} \cdot \hat{\mathbf{V}}'^i$

(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 $5N_\mathrm{k}N_\mathrm{C}$ convolutions required in the spatial domain with simple element-wise multiplications, while adding only $4N_\mathrm{k}$ Fourier transforms. The other operations are equivalent in both domains.

the time-reversal property of the Fourier transform and the reality condition $\hat{f}(-\xi) = \overline{\hat{f}(\xi)}$. Using the aforementioned mappings, equations ((4.3)) to ((4.2)) can be rewritten as

$$\Delta\hat{\mathbf{W}}^{ij} = \hat{\mathbf{V}}^i \cdot \overline{\hat{\mathbf{h}}^j} - \hat{\mathbf{V}}'^i \cdot \overline{\hat{\mathbf{h}}'^j} \tag{3.5}$$

$$\mathbb{E}[\hat{V}^i_{xy} \mid \hat{\mathbf{h}}] = \begin{cases} \sum_j \hat{W}^{ij}_{xy} \hat{h}^j_{xy} + N_\mathrm{V}^2 b_i & \text{for } x, y = 0 \\ \sum_j \hat{W}^{ij}_{xy} \hat{h}^j_{xy} & \text{for } x, y \neq 0 \end{cases} \tag{3.6}$$

$$\hat{\mathbf{h}}^j \sim \mathrm{F}\left(\max(0, \mathrm{F}^{-1}(\hat{\mu}^j) + c_j + \mathcal{N}(0, \sigma^2))\right) \tag{3.7}$$

$$\hat{\mu}^j = \sum_i \overline{\hat{\mathbf{W}}^{ij}} \cdot \hat{\mathbf{V}}^i \tag{3.8}$$

where $\sigma^2 = \mathrm{sigm}(\mathrm{F}^{-1}(\hat{\mu}^j) + c_j)$, $\hat{x} = \mathrm{F}(x)$ denotes x in the frequency domain, 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 3.1b.

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 $2N_\mathrm{k}$ 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.

3.1.3 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 the calculation of the element-wise operations, which can be performed in parallel. Thus we distribute the processing of a single 2D image over $N_\mathrm{V}(\lfloor N_\mathrm{V}/2 \rfloor + 1) \times N_\mathrm{C}$ 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 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

$$4N_{\mathrm{v}}^2 N_{\mathrm{c}} + \frac{4N_{\mathrm{v}}^2 N_{\mathrm{k}}}{s^2} + 4N_{\mathrm{v}}^2 N_{\mathrm{k}} N_{\mathrm{c}} \tag{3.9}$$

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

$$4N_v^2 N_b N_c + \frac{4N_v^2 N_b N_k}{s^2} + 4N_w^2 N_k N_c \tag{3.10}$$

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 to speed 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

$$4N_v^2 N_b N_c + 4N_v^2 N_b N_k + 4N_v^2 N_k N_c \tag{3.11}$$

Table 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 image at a time rather than a batch, and still outperforms batched learning in the spatial domain in terms of speed (see Section 3), despite not using batches.

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.

Layer	N_b	N_v	N_c	N_w	N_k	s	Memory in MB		
							Freq	Spat	B-FFT
1	128	224	3	11	48	4	28.7	147.1	—
2	128	55	48	5	128	1	72.9	260.5	330.9
3	128	27	128	3	192	1	69.2	114.8	182.3
4	128	13	192	3	192	1	24.0	33.0	55.5
5	128	13	192	3	128	1	16.1	27.3	42.3

3.1.4 Mapping Strided Convolutional to Stride-1 Convolutions

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. This reduces the number of hidden units per channel to $N_\mathrm{h} = N_\mathrm{v}/s$, hence significantly reducing training times and memory required for storing the hidden units during training. The energy of a strided convolutional RBM (sconvRBM) is defined as

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=0}^{N_\mathrm{c}-1}\sum_{j=0}^{N_\mathrm{k}-1}\sum_{x,y=0}^{N_\mathrm{h}-1}\sum_{u,v=-\lfloor N_\mathrm{w}/2\rfloor}^{\lfloor (N_\mathrm{w}-1)/2\rfloor} h_{xy}^{j} w_{uv}^{ij} v_{sx+u,sy+v}^{i} - \sum_{i=0}^{N_\mathrm{c}-1} b_i \sum_{x,y=0}^{N_\mathrm{v}-1} v_{xy}^{i} - \sum_{j=0}^{N_\mathrm{k}-1} c_j \sum_{x,y=0}^{N_\mathrm{h}-1} h_{xy}^{j} \tag{3.12}$$

Convolutions with a stride $s > 1$ can be expressed equivalently as convolutions with stride $s = 1$ by reorganizing the values of $\mathbf{v}^i$ and $\mathbf{w}^{ij}$ to $\mathbf{V}^{i'}$ and $\mathbf{W}^{i'j}$ as illustrated in Figure 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

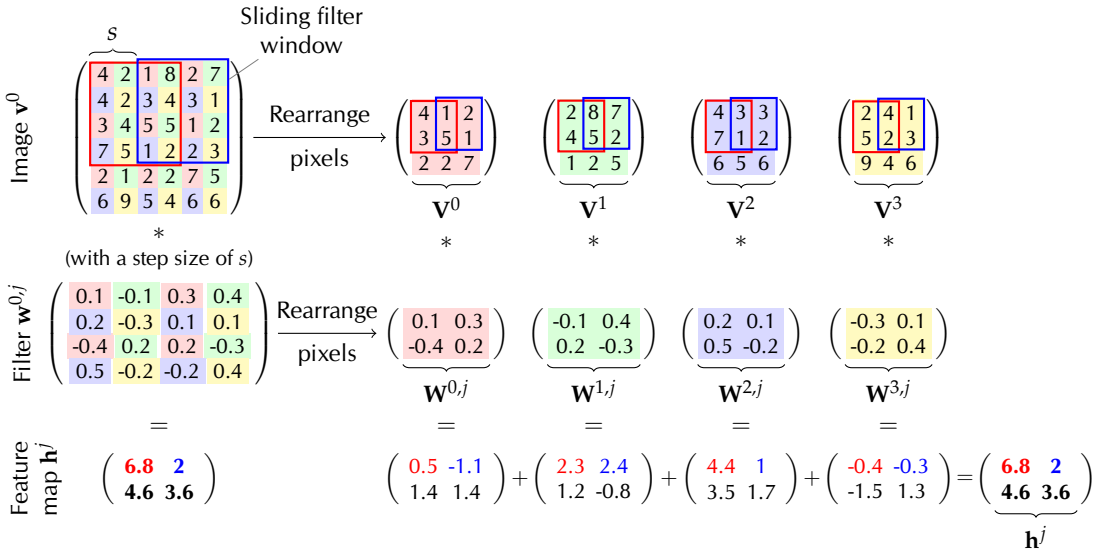


Figure 3.2: Illustration of convolutions with a sliding window step size $s = 2$ as used during filtering in a sconvDBN. A convolution of a given stride size (left side) can be efficiently calculated as the sum of multiple individual convolutions with $s = 1$ (right side) after rearranging the pixels 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 color.

domain. The new indices of $V^{i'}_{x'y'}$ and $W^{i'j}_{u'v'}$ can be calculated from the old indices of v^i_{xy} and w^{ij}_{uv} as follows:

$$x' = \lfloor x/s \rfloor \qquad u' = \lfloor u/s \rfloor \tag{3.13}$$

$$y' = \lfloor y/s \rfloor \qquad v' = \lfloor v/s \rfloor \tag{3.14}$$

$$i' = s^2 i + s(y \bmod s) + (x \bmod s) \tag{3.15}$$

After reorganizing $\mathbf{v}^i$ and $\mathbf{w}^{ij}$ to $\mathbf{V}^{i'}$ and $\mathbf{W}^{i'j}$, the energy of the model can be rewritten as

$$E(\mathbf{V},\mathbf{h}) = -\sum_{i=0}^{N_\mathrm{C}-1}\sum_{j=0}^{N_\mathrm{k}-1}\sum_{x,y=0}^{N_\mathrm{h}-1}\sum_{u,v=-\lfloor N_\mathrm{W}/2\rfloor}^{\lfloor (N_\mathrm{W}-1)/2\rfloor} h^j_{xy} W^{ij}_{uv} V^i_{x+u,y+v} - \sum_{i=0}^{N_\mathrm{C}-1} b_i \sum_{x,y=0}^{N_\mathrm{V}-1} V^i_{xy} - \sum_{j=0}^{N_\mathrm{k}-1} c_j \sum_{x,y=0}^{N_\mathrm{h}-1} h^j_{xy} \tag{3.16}$$

$$= -\sum_{i=0}^{N_\mathrm{C}-1}\sum_{j=0}^{N_\mathrm{k}-1} \mathbf{h}^j \bullet (\tilde{\mathbf{W}}^{ij} * \mathbf{V}^i) - \sum_{i=0}^{N_\mathrm{C}-1} b_i \sum_{x,y=0}^{N_\mathrm{V}-1} V^i_{xy} - \sum_{j=0}^{N_\mathrm{k}-1} c_j \sum_{x,y=0}^{N_\mathrm{h}-1} h^j_{xy} \tag{3.17}$$

where $*$ denotes periodic convolution. The number of channels, number of visible units per channel and number of weights per channel after reorganization are given by $N_\mathrm{C} = N_\mathrm{c} * s^2$, $N_\mathrm{V}^2 = N_\mathrm{v}^2/s^2$ and $N_\mathrm{W}^2 = N_\mathrm{w}^2/s^2$, respectively.

3.2 Evaluation of Runtime

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

Table 3.2: Training parameters.

Parameter	ImageNet (2D)		OASIS (3D)	
	1st layer	2nd layer	1st layer	2nd layer
Filter size	5 to 52	5 to 13	3 to 36	3 to 9
Stride size	1, 2, 4	1	1, 2, 4	1
Number of channels	3	16, 32, 64	1	8, 16, 32
Number of filters	32	32	32	16
Image dimension	512^2	128^2	128^3	64^3
Number of images	256	256	100	100
Batch size	128	128	25	25

Table 3.3: Hardware specification of our test system.

Processor	Intel i7-3770 CPU @ 3.40 GHz
CPU Memory	8 GB
Graphics Card	NVIDIA GeForce GTX 660
GPU Cores	960 cores @ 1.03 GHz
GPU Memory	2 GB

in Table 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 included in this paper. The hardware details of our test environment are summarized in Table Table 3.3.

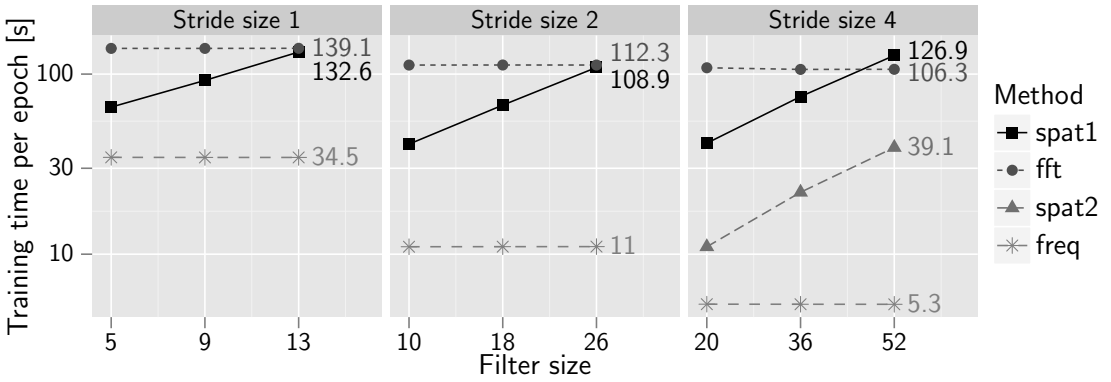
For the comparison on 2D images, we used a dataset of 256 natural color images from the ImageNet dataset (Deng et al., 2009). All images were resampled to a resolution

of 512×512 pixels per color channel. For the evaluation on 3D images, we used 100 magnetic resonance images (MRIs) of the brain from the OASIS dataset (Marcus et al., 2007). We resampled all volumes to a resolution of $128 \times 128 \times 128$ voxels and a voxel size of $2 \times 2 \times 2\,\text{mm}$.

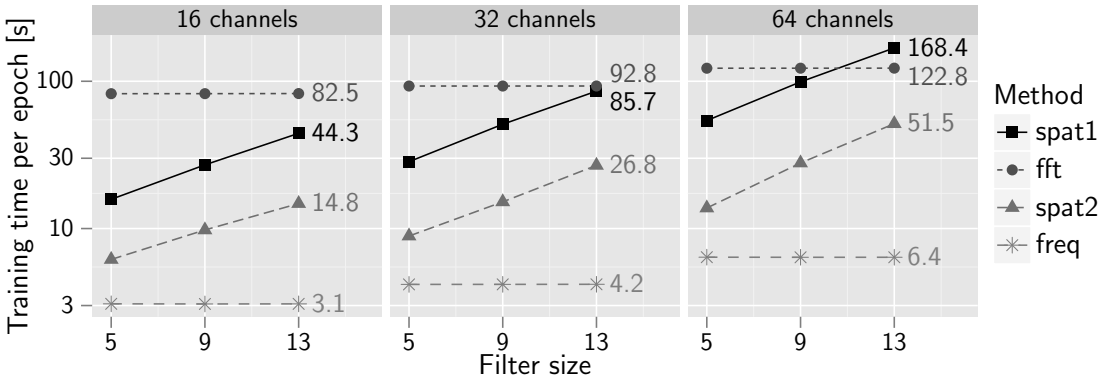
3.2.1 Running Time Analysis on 2D Color Images (ImageNet)

Figure 3.3a 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 512×512 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. For a stride of 1, 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 convolutions. 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 3.3b shows a similar comparison for training the second convRBM layer for a stride size of 1 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



(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).

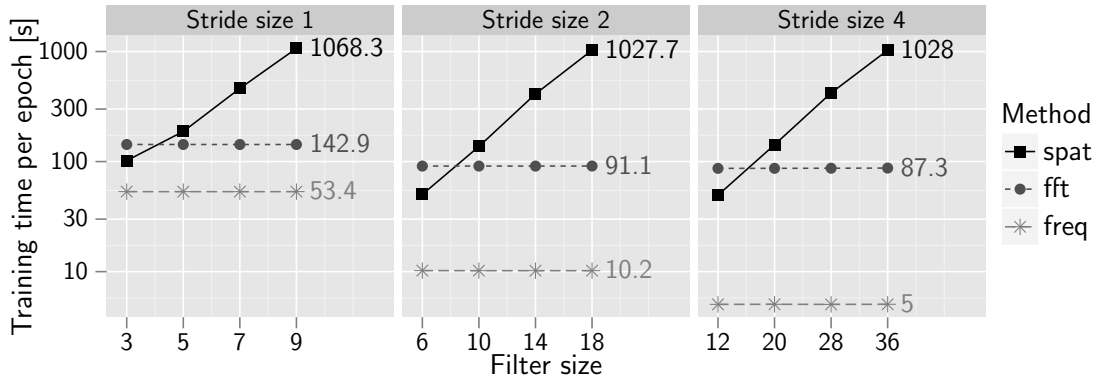
Figure 3.3: Comparison of running times for training a (a) first and (b) second layer sconvRBM on 2D images using our frequency domain method (freq) and three alternative methods using different convolution implementations: single image convolutions (spat1), batched convolutions (spat2), and convolution 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.

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

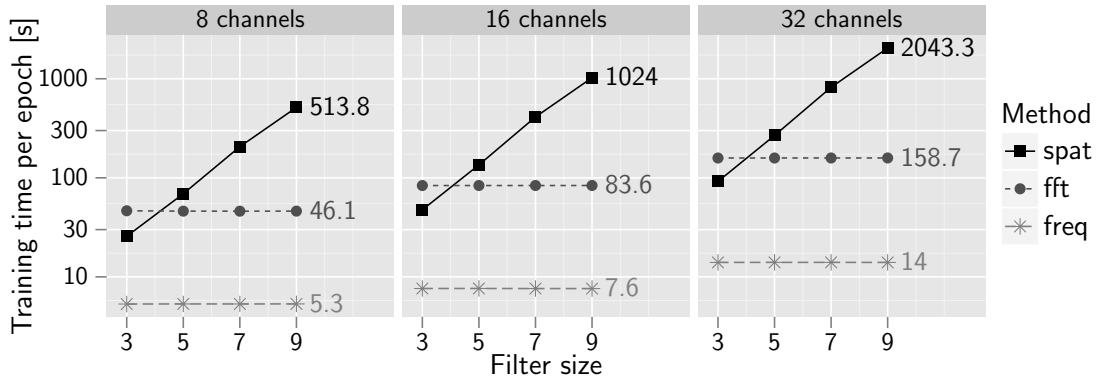
than batched training and training in the frequency domain. To summarize, training of 2D images in the frequency domain is much faster than training in the spatial domain even for small filter sizes. Using the largest filter kernels in both layers, the proposed method is shown to yield a speedup of 7 to 8 times compared to state-of-the-art GPU implementations.

3.2.2 Running Time Analysis on 3D Volumes (OASIS)

Figure 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 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 5 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 1 depend strongly on the time required to calculate the expectation of the hidden units and to sample the hidden units. Hence, 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.



(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 8, 16, and 32 channels (stride size 1).

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).

4 Modeling the Variability of Brain MRIs

4.1 Variability and Manifold Learning

The need for manifold learning often arises when very high-dimensional data needs to be analyzed but the intrinsic dimensionality of the data is much lower. This situations occurs, for example, when trying to visualize variability and common patterns in a given group of magnetic resonance images (MRIs) of the brain. Each image can be 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.

Various methods for manifold learning have been proposed (e.g., locally linear embedding (LLE) (Saul and Roweis, 2003), Laplacian eigenmaps (LEM) (Belkin and Niyogi, 2003), Isomaps (Tenenbaum et al., 2000)), with Isomaps and LEM being the most popular for medical image analysis. Both methods require a prebuild proximity graph. In order to build the proximity graph, it is assumed that the manifold space is locally linear, which

means that distances between neighboring points in manifold space can be approximated by their distances in ambient space. Gerber et. al. have shown that the choice of a suitable distance measure is crucial for manifold learning using Isomaps and that the warping distance between brain images improves the learning performance over previously used Euclidean distances in the image space (Gerber et al., 2010).

Manifolds have been used to regularize the segmentation of brain ventricles (Etyngier et al., 2007), and to constrain the deformable registration of brain images to have biologically plausible parameters (Hamm et al., 2010). Gerber et al. used Isomaps to predict clinical parameters of Alzheimers's (AD) patients (Gerber et al., 2010), and Wolz et al. used Laplacian eigenmaps to perform biomarker discovery (Wolz et al., 2011), also of AD patients, and atlas propagation for the segmentation of the hippocampus (Wolz et al., 2010a).

Other application in the area of MS. Multiple sclerosis (MS) is an inflammatory and degenerative 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 white matter on conventional MRI, and the death of nervous tissue leading to global and regional atrophy. A number of imaging biomarkers, such as lesion volume and whole brain volume, 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. Methods based on statistics of diffeomorphisms have been used to discover patterns of regional atrophy Ceccarelli et al. (2012), but they are not designed to directly model morphological variability. 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 modeling of

brain morphology and lesion distribution would further our knowledge of how these two key pathological features interact. However, this type of modeling 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 representations of medical images Aljabar et al. (2011); Wolz et al. (2012, 2010b). Manifold learning is motivated by the assumption that the space of brain images can be modeled to some approximation by a low-dimensional manifold. Various methods for manifold learning have been proposed (e.g., locally linear embedding (LLE), Laplacian eigenmaps (LEM), Isomaps) Cayton (2005), with Isomaps and LEM being the most popular for medical image analysis. Most such methods require a prebuilt proximity graph based on a selected distance measure, the choice of which can be crucial for manifold learning **?**. Aljabar et al. proposed a method Aljabar et al. (2011) for the joint modeling of multiple image features of neonatal brains. First, separate manifolds based on features from geometric surface models, non-rigid deformations, and image intensities are learned, then a second dimensionality reduction step is used to combine the individual manifold parameters.

4.2 Manifold of AD Patients

In this paper, we propose a novel approach for learning the manifold of brain images that uses a deep belief network (DBN) Hinton et al. (2006) to discover patterns of similarity in groups of images. In contrast to previous brain manifold learning algorithms, 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.

- DBNs mostly limited to 2D images, still the case?

- Our fast training method allows DBNs to be used for manifold learning of 3D medical images

Contribution is the demonstration that DBNs can learn a low-dimensional manifold of brain volumes that detects modes of variation that correlate to demographic and disease parameters.

4.2.1 Method

We have evaluated the proposed method on a subset of the ADNI dataset Petersen et al. (2010), containing 300 T1-weighted MRIs of Alzheimer's disease (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 $2.0 \times 2.0 \times 2.0\,\text{mm}$. 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 dataset was divided into a training set and a test set such that each set contains 75 AD and 75 normal subjects. To learn the manifold of brain MRIs, we used a DBN with three convRBM layers and two RBM layers. After three convRBMs, the dimension of each image is reduced to $8 \times 8 \times 8$ and small enough for RBMs. The training of the DBN took approximately 43 hours on two GeForce GTX 560 Ti graphics cards.

Our proposed method performs manifold learning by reducing the dimensionality of the input images using a DBN, a deep generative model that is composed of multiple restricted Boltzmann machines (RBMs) Hinton et al. (2006) as illustrated by the simplified example in Fig. **??**. An RBM is a Markov random field with trainable weights whose nodes

are divided into a visible layer **v** representing the inputs of the model and a hidden layer **h** representing extracted features from the inputs. The first RBM receives the intensity values of a group of images 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. 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 convolutional RBMs (convRBMs), a type of RBM that uses weight sharing to reduce the number of trainable weights, albeit at the cost of using the much more computationally expensive convolutions instead of multiplications. In the remainder of this section, we will briefly review the training of convRBMs, followed by a description of our novel training algorithm that performs parameter learning in frequency domain. For comprehensive introductions to RBMs and convRBMs, the reader is referred to Hinton et al. (2006) and **?**, respectively.

Our algorithm for the dimensionality reduction of an input image using a convRBM is illustrated in Fig. **??**. In contrast to previous work that uses max pooling to reduce the dimensionality **?**, all steps involved in our method are invertible, which allows the reconstruction of images from their manifold coordinates. In the first step, the pixels of an input image are reorganized into multiple images of lower resolution in order to reduce the dimensionality of a single image. The number of images in the image vector is then reduced with the following steps. To apply the model to real-valued data like the intensities of some medical images, the visible units are modeled as Gaussian units.

When the visible units have been mean centered and standardized to unit variance, the expectation of the visible units is given by

$$\mathbb{E}[v_i \mid \mathbf{h}] = \sum_j w_{ij} * h_j + b_i \tag{4.1}$$

where $v_i, h_j, b_i \in \mathbb{R}^{N \times N \times N}$, b_i are bias terms, N is the image size, $w_{ij} \in \mathbb{R}^{N_w \times N_w \times N_w}$ are the weights, N_w is the size of a weight kernel, and $*$ denotes circular 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 as the hidden units, which has shown to improve the learning performance **?** of RBMs. The hidden units can be sampled with

$$h_j \sim \max(0, \mu_j + \mathcal{N}(0, \text{sigm}(\mu_j))) \tag{4.2}$$

where $\mu_j = \sum_i \tilde{w}_{ij} * v_i + c_j$, $c_j \in \mathbb{R}^{N \times N \times N}$, $\tilde{w}$ denotes a flipped version of w in all three dimensions, $\mathcal{N}(0, \sigma^2)$ denotes Gaussian noise, and $\text{sigm}(x)$ is the sigmoid function defined as $\text{sigm}(x) = (1 + \exp(-x))^{-1}, x \in \mathbb{R}$. The weights and bias terms of a convRBM can be learned using contrastive divergence (CD) Hinton et al. (2006). During each iteration of the algorithm, the gradient of the parameters is estimated and a gradient step with a fixed learning rate is performed. The gradient of the weights can be approximated by:

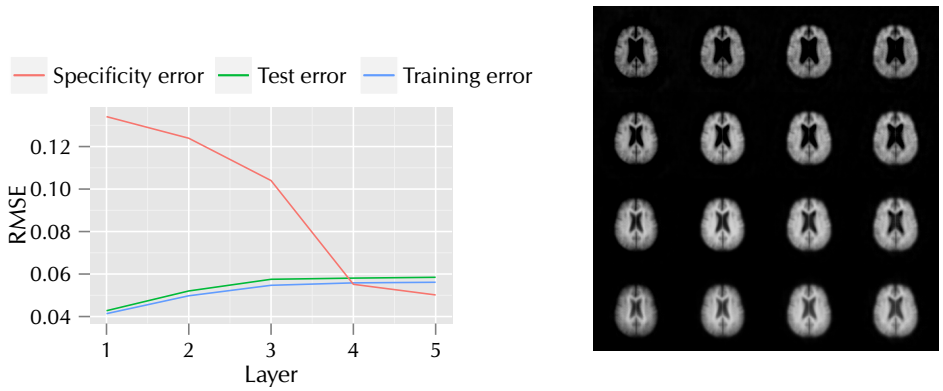
$$\Delta w_{ij} = v_i * \tilde{h}_j - v_i' * \tilde{h}_j' \tag{4.3}$$

where h_j and h_j' are samples drawn from $p(h_j \mid \mathbf{v})$ and $p(h_j \mid \mathbf{v}')$ using ((4.2)) and $v_i' = \mathbb{E}[v_i \mid \mathbf{h}]$.

4.2.2 Evaluation

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 as the average root mean squared error (RMSE) between an image and its reconstruction, normalized by the intensity range of the input image. The specificity was measured by calculating the average RMSE between images randomly generated from the manifold model and the most similar images from the training set. Figure Figure 4.1(a) 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 Figure 4.1(b) 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 (from left to right) appear to increase in brain size, and the images sampled along the second manifold dimension (from bottom to top) appear to increase in ventricle size. Figure Figure 4.2 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



(a) Generalizability vs. specificity

(b) Generated images

Figure 4.1: (a) The similarity of the reconstruction errors between the training and test images indicates that no overfitting occurs. The opposite slopes of the reconstruction errors and error of generated images (specificity error) indicates a trade-off between generalizability vs. specificity in the earlier phases of training, but the low errors at Layer 5 indicate that the method is both generalizable and specific. (b) 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.

with an increase of the second manifold coordinate. The AD/normal status is indicated by the frame color of each image. The vertical separation between AD and normals suggests that the second manifold coordinate is potentially of practical use in differentiating between AD and normal.

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 (mini-mental state examination (MMSE) score, AD/normal status) with the manifold coordinates (x_1 and x_2). The results of the correlation tests are summarized in Table Table 4.1. Age, MMSE and AD/normal status show highly significant correlations with x_2 (p-values between 9.85×10^{-9} and 3.53×10^{-7}), which makes intuitive sense because x_2 visually correlates with ventricle size. The first manifold coordinate correlates strongest with gender (p-value $= 8.24 \times 10^{-9}$), which also

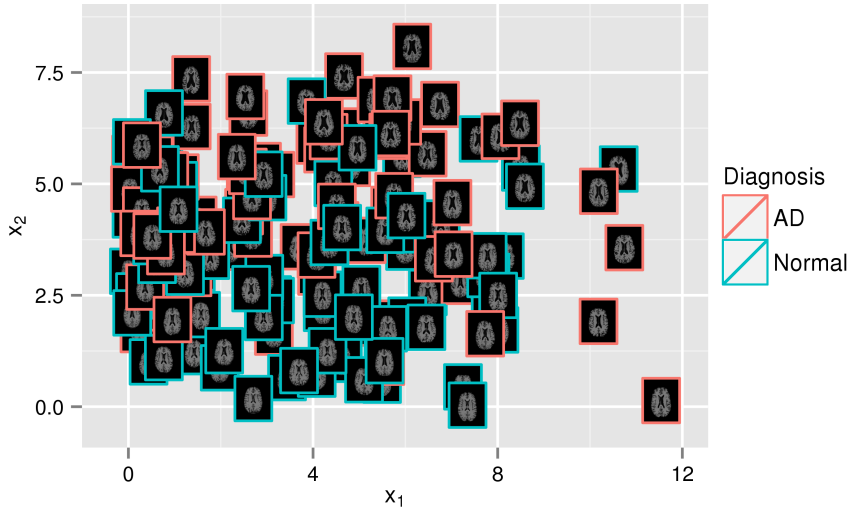


Figure 4.2: Axial slices of volumes from the training set plotted against their manifold coordinates. The brains with larger ventricles, indicative of atrophy, are mostly at the top, which is also where most of the AD patients are.

Table 4.1: Pearson correlation r of demographic and clinical parameters with manifold coordinates (x_1, x_2). The stronger correlation in each column is highlighted in bold.

	Age		Gender		MMSE		AD/Normal Status	
	r	p-value	r	p-value	r	p-value	r	p-value
x_1	-0.17	0.03	**0.45**	$\mathbf{8.24{\cdot}10^{-9}}$	0.01	0.89	-0.03	0.69
x_2	**0.45**	$\mathbf{9.85{\cdot}10^{-9}}$	0.19	0.02	**−0.40**	$\mathbf{3.53{\cdot}10^{-7}}$	**0.41**	$\mathbf{1.54{\cdot}10^{-7}}$

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.

4.3 Variability of Morphology and Lesion Distribution

We present a new method for modeling the variability in the morphology and lesion distribution of a large set of MRIs of MS patients. Our method is built using a deep belief

network (DBN) Hinton et al. (2006), a layered network whose parameters can be learned from training images. An advantage of DBNs over other manifold learning methods is that it does not require a prebuilt proximity graph, which is particularly beneficial for modeling lesions, because the spareness 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 modeling 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.

4.3.1 Method

Our proposed model for pattern discovery consists of three main components (a) a model that aims to find patterns of morphological changes in deformation fields, (b) a model that aims to find patterns in the spatial distribution of lesions, and (c) a joint model that aims to find concurring deformation and lesion distribution patterns. The morphology model is learned from a set of displacement fields that are calculated via non-rigid registration from a set of T1-weighted (T1w) brain MRIs $D \subset I$, $I = \{I_n \mid I_n \colon \Omega \mapsto \mathbb{R}\}$, $\Omega \subset \mathbb{R}^3$ 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 FLIRT Jenkinson et al. (2002). Then for each image $I_n \in D$, the displacement field

u_n, $u\colon \Omega \mapsto \mathbb{R}^3$, that describes the non-rigid transformation from template coordinates to image coordinates is calculated using 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 $p(u \mid D_1, \dots)$ 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 $p(u \mid D_1, \dots)$, where the parameters $(D_1, \dots)^T$ 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 $\mathbb{E}[D_1, \dots \mid u_1]$ and $\mathbb{E}[D_1, \dots \mid u_2]$. Furthermore, we can visualize the modes of morphological variability of MS brain images, by sampling the space of displacement fields spanned by $(D_1, \dots)^T$ by calculating the expectation $\mathbb{E}[u \mid D_1, \dots]$ for a range of values for $(D_1, \dots)^T$.

The probability density function $p(u)$ is modeled by a deep belief network (DBN) Hinton et al. (2006), a generative probabilistic graphical model consisting of one layer of observed random variables (visible units) and multiple layers of latent random variables (hidden units). In a DBN, each pair of adjacent layers of random variables form a restricted Boltzmann machine (RBM). The first RBM 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

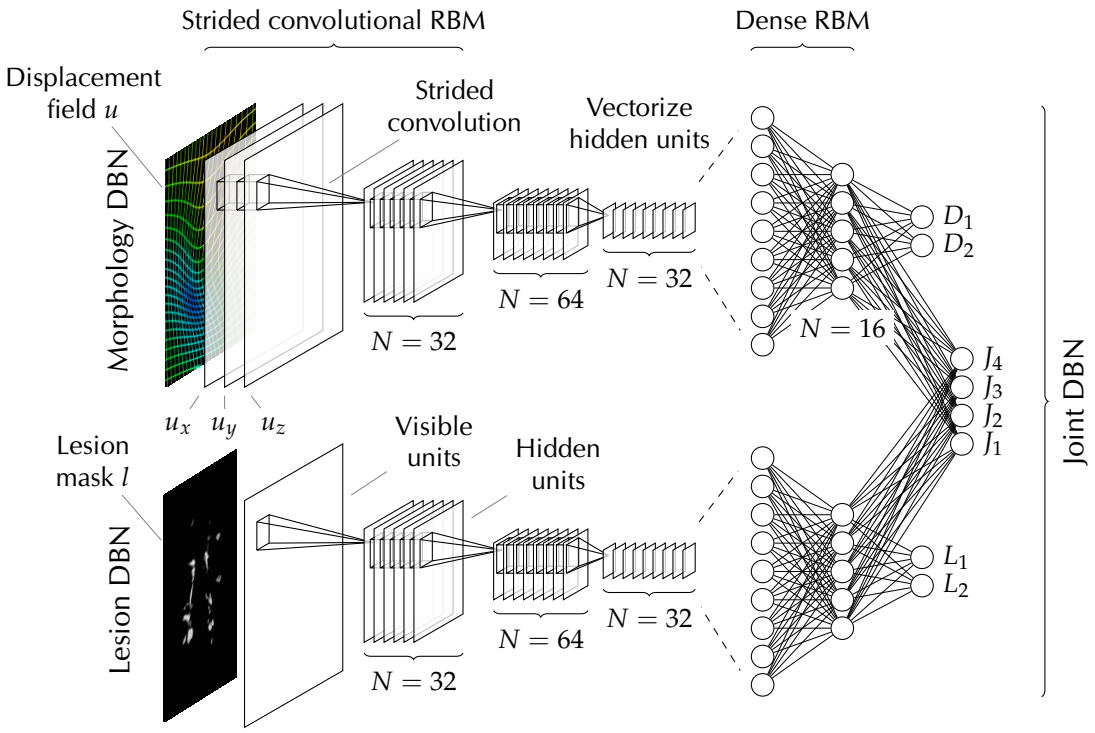


Figure 4.3: DBN models used for discovering patterns.

three strided convolutional RBMs[1] (sconvRBMs) and two dense RBMs **?** with 16 and 2 hidden units, respectively. In the following, we will briefly review the sconvRBM, as this model is less often described in the literature, followed by how the visible and hidden units of the DBN relate to displacement fields and displacement field distribution parameters. A sconvRBM is a type of RBM in which the probabilistic relationships between the visible and hidden units are expressed in terms of strided convolutions, a type of convolution that shifts the filter kernel as a sliding window with a step size or stride $s > 1$. Due to the much smaller number of trainable parameters compared to dense RBMs, sconvRBMs are best

[1]The three sconvRBMs have stride sizes of $2 \times 2 \times 1$, $2 \times 2 \times 2$, $1 \times 1 \times 1$, filter sizes of $10 \times 10 \times 7$, $10 \times 10 \times 10$, $3 \times 5 \times 3$, and 32, 64, 32 filters, respectively.

suited for learning low- to mid-level features from very high-dimensional data. Compared to other more commonly used convolution-based RBMs **?**, 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. The complete morphology DBN can be trained layer-by-layer by training each RBM individually using contrastive divergence Hinton et al. (2006). Once the parameters of the DBN have been learned from training data, we can use the model for inference. Let $\mathbf{v}_{\mathrm{d},1}, \ldots, \mathbf{v}_{\mathrm{d},5}$, $\mathbf{h}_{\mathrm{d},1}, \ldots, \mathbf{h}_{\mathrm{d},5}$, and $\boldsymbol{\theta}_{\mathrm{d},1}, \ldots, \boldsymbol{\theta}_{\mathrm{d},5}$ 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 $u \sim p(u \mid D_1, \ldots)$ with

$$(D_1, \ldots)^T = \mathbb{E}[D_1, \ldots \mid u_n] = \mathbb{E}[\mathbf{h} \mid \mathbf{v}_{\mathrm{d},5}, \boldsymbol{\theta}_{\mathrm{d},5}] \tag{4.4}$$

$$\mathbf{v}_{\mathrm{d},i+1} = \mathbb{E}[\mathbf{h} \mid \mathbf{v}_{\mathrm{d},i}, \boldsymbol{\theta}_{\mathrm{d},i}] \tag{4.5}$$

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

$$\bar{u} = \mathbb{E}[u \mid D_1, \ldots] = \mathbb{E}[\mathbf{v} \mid \mathbf{h}_{\mathrm{d},1}, \boldsymbol{\theta}_{\mathrm{d},1}] \tag{4.6}$$

$$\mathbf{h}_{\mathrm{d},i} = \mathbb{E}[\mathbf{v} \mid \mathbf{h}_{\mathrm{d},i+1}, \boldsymbol{\theta}_{\mathrm{d},i+1}] \tag{4.7}$$

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

The input into our lesion model is a set of 3D binary lesion masks $l_n \in I$, which have been created from T2-weighted (T2w) and PD-weighted (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 $l_n \sim p(l \mid L_1, \dots)$ that can be parameterized by only relatively few parameters $(L_1, \dots)^T$. The task of finding lesion patterns is to discover the underlying probability density function $p(l \mid L_1, \dots)$, where the parameters $(L_1, \dots)^T$ represent patterns of variability of MS lesions. Similar to the morphology DBN, the lesion DBN consists of three sconvRBMs[2] and two dense RBMs with 16 and 2 hidden units, respectively. However, we modified the energy function of the sconvRBMs to account for the sparse activation of MS lesion masks. Large black regions without local structure can lead to random activations of the hidden units and consequently the learning of random filters. To overcome this problem, we propose to incorporate a region of interest (ROI) term into the energy equation of the sconvRBM, which allows constraining the filter learning process to a given ROI. This can be achieved by 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 ROI was chosen to include 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 $(L_1, \dots)^T$ of $l_n \sim p(l \mid L_1, \dots)$ in the same manner as in ((4.4)) and ((4.5)). Likewise, the mean lesion configuration $\bar{l}$ given the distribution parameters $(L_1, \dots)^T$ can be calculated in the same manner as in ((4.6)) and ((4.7)).

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 $p(u, l \mid J_1, \dots)$. The joint DBN consists of two pathways, each corresponding

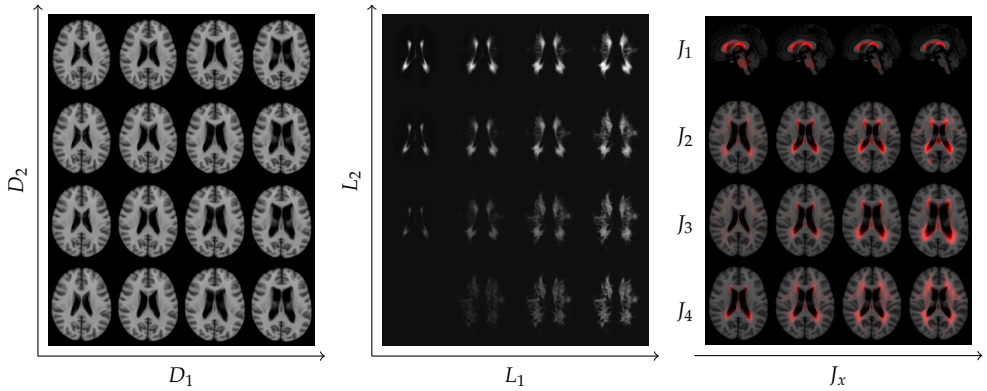
[2]The three sconvRBMs have stride sizes of $4 \times 4 \times 2$, $2 \times 2 \times 2$, $2 \times 2 \times 2$, filter sizes of $20 \times 20 \times 10$, $14 \times 14 \times 10$, $10 \times 14 \times 6$, and 32, 64, 64 filters, respectively.

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 $p(\mathbf{v}_\mathrm{j}, \mathbf{h}_\mathrm{j} \mid \boldsymbol{\theta}_\mathrm{j})$, where $\mathbf{v}_\mathrm{j} = (\mathbf{h}_{\mathrm{d},4}^T, \mathbf{h}_{\mathrm{l},4}^T)^T$ and $\mathbf{h}_\mathrm{j} = (J_1, \dots)^T$ are the modes of variability of morphological and lesion distribution changes.

4.3.2 Evaluation

The proposed method was evaluated on a dataset from an MS clinical trial of 474 secondary progressive MS patients. For each subject, the dataset contains one T1w, T2w, and PDw MRI with a resolution of $256 \times 256 \times 50$ voxels and a voxel size of $0.937 \times 0.937 \times 3.000\,\mathrm{mm}$. The main preprocessing steps included rigid registration, brain extraction, intensity normalization, and background cropping. We then trained the 3 DBN models as described in Sect. **??**.

The invertibility of our model allows the reconstruction of images from the distribution parameters to visualize the discovered patterns of variability. Figure 4.4a shows axial 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 4.4b shows axial slices from the mean lesion configurations $\mathbb{E}[l \mid L_1, L_2]$ 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.



(a) Morphology manifold (b) Lesion manifold (c) Joint manifold

Figure 4.4: 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.

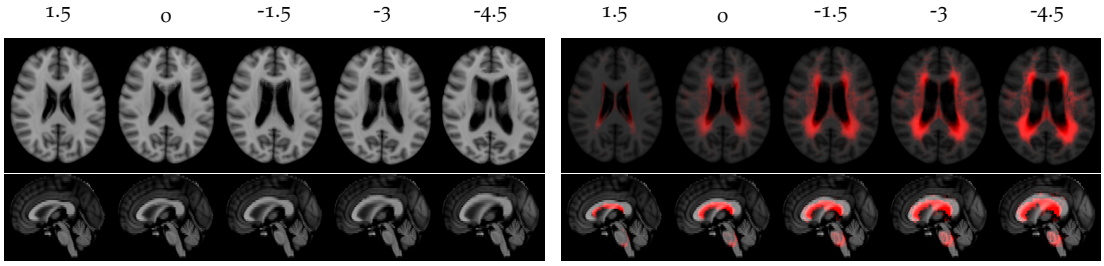
To visualize concurring patterns of morphology and lesion distribution, we sampled images from the joint model $p(u, l \mid J_1, \ldots, J_4)$ as shown in Fig. 4.4c. The images are deformed template images with superimposed lesion masks. For each row, we varied only one distribution parameter and set the remaining parameters to their mean values. Of the 4 parameters, J_3 visually corresponds most closely to the "classic" 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.

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) Fischer et al. (1999) and its components (Timed 25-Foot Walk, T25W;

Table 4.2: 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 (* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$).

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

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 Table 4.2. In the individual models, all parameters correlate significantly with 9-HPT, PASAT, and MSFC, except for D_2 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 modeled by J_1 occur in the pons, which is known to be of clinical significance for mobility. Overall, the learned parameters correlate better than the established imaging biomarkers.



(a) Morphological changes (b) Lesion changes

Figure 4.5: Axial (top row) and mid-sagittal (bottom row) slices of volumes representing the spectrum of MSFC scores from 1.5 to -4.5. A decrease in MSFC shows the classic pattern of MS pathology.

Another benefit of our model is the ability to visualize the progression of a "mean" secondary progressive MS brain along a range of MSFC scores. To demonstrate, we trained 4 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 Figure 4.5 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.

4.4 Discussion

Potentially follow up with a discussion where I can discuss related approaches that came afterwards and what needs to be done.

5 Medical Image Segmentation

- Start with general MS lesion segmentation literature review.
- What are the challenges and how could deep learning help to overcome them?
- Then literature using deep learning for segmentation. Patch-based approaches for lesion segmentation, even if not MS lesions.
- More recent approach are fully convolutional. Then explain my method including the evaluation.

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.

5.1 Related Work

5.1.1 MS Lesion Segmentation Methods

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. Unsupervised methods do not require a labeled data set for training. Instead, lesions are identified as an outlier class using, e.g., clustering methods (Shiee et al., 2010; Souplet et al., 2008) or generative models (Weiss et al., 2013). In addition to modelling MS lesions as a separate cluster, Lesion-TOADS (Shiee et al., 2010) employs a topological and a statistical atlas to produce a topology-preserving segmentation. Current supervised approaches typically start with a large set of features, either predefined by the user (Geremia et al., 2010) or gathered in a feature extraction step such as by deep learning(Yoo et al., 2014), which is followed by a separate training step with labeled data to determine which set of features are the most important for segmentation in the particular domain. A recent breakthrough for automatic segmentation using deep learning comes from the domain of cell membrane segmentation, in which Ciresan et al. (Cireşan et al., 2012) proposed to classify the centers 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.

5.1.2 Patch-based approaches using Deep Learning

5.1.3 Fully Convolutional Approaches

Recently, different CNN architectures (Brosch et al., 2015; Kang and Wang, 2014; 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 scales up more efficiently with image resolution. Kang et al. introduced the fully convolutional neural network (fCNN) for the segmentation of crowds in surveillance videos (Kang and Wang, 2014). 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. Ronneberger et al. (Ronneberger et al., 2015) proposed an 11-layer u-shaped network architecture called u-net, 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.

5.2 Method

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 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 network uses deconvolution instead of upsampling in the expanding pathway and predicts segmentations that have the same resolution as the input images and therefore does not require special handling of the border regions.

5.2.1 Solving Segmentation as an Optimization Problem

In this paper, the task of segmenting MS lesions is defined as finding a function s that maps multi-modal images I, e.g., $I = (I_{\text{FLAIR}}, I_{\text{T1}})$, to corresponding lesion masks S. 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

$$\hat{s} = \arg\min_{s \in \mathcal{S}} \sum_n E(S_n, s(I_n)), \tag{5.1}$$

where $\mathcal{S}$ is the set of possible segmentation functions, and E is an error measure that calculates the dissimilarity between ground truth segmentations and predicted segmentations.

5.2.2 Model Architecture

The set of possible segmentation functions, $\mathcal{S}$, is modeled by the convolutional encoder network with shortcut connections (CEN-s) illustrated in Fig. Figure 5.1. A CEN-s is a type of convolutional neural network (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 $x_i^{(0)}(\mathbf{p})$, $i \in [1, C]$, where i indexes the modality or input channel, C is the number of modalities or channels, and $\mathbf{p} \in \mathbb{N}^3$ 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

$$x_j^{(l)} = \max\left(0, \sum_{i=1}^{C} \tilde{w}_{c,ij}^{(l)} * x_i^{(l-1)} + b_j^{(l)}\right), \tag{5.2}$$

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 $w_{c,ij}^{(l)}$, 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. 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

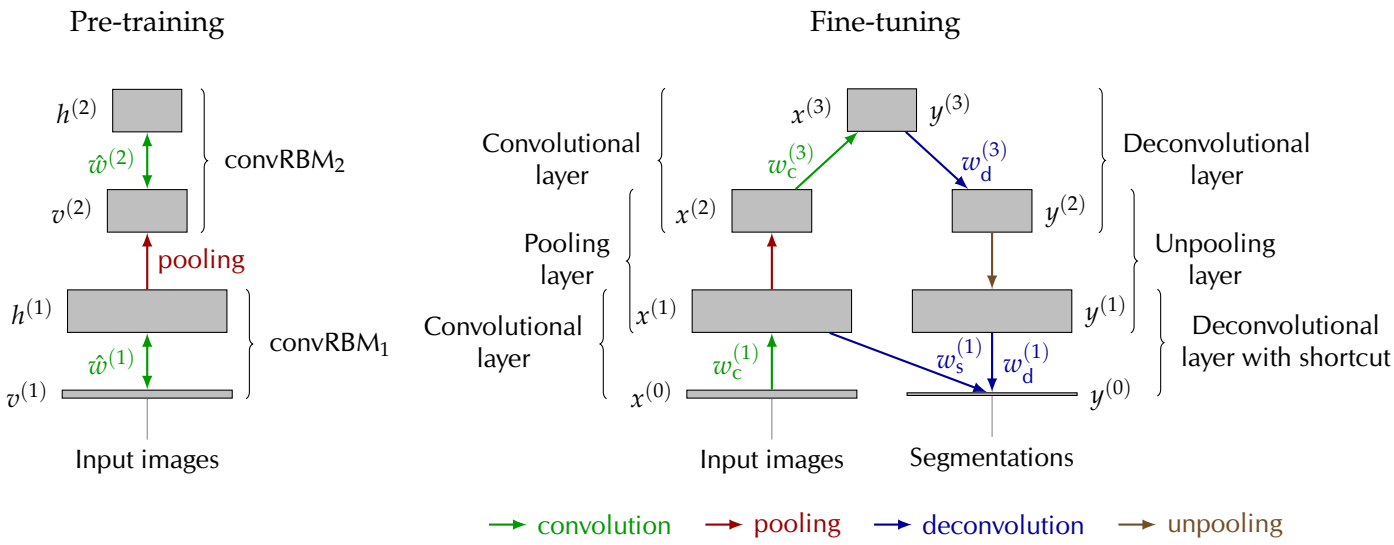


Figure 5.1: Pre-training and fine-tuning of the 7-layer convolutional encoder network with shortcut that we used for our experiments. Pre-training is performed on the input images using a stack of convolutional RBMs. The pre-trained weights and bias terms are used to initialize a convolutional encoder network, which is fine-tuned on pairs of input images, $x^{(0)}$, and segmentations, $y^{(0)}$.

deconvolutional layer uses the extracted features of the convolutional pathway to calculate abstract segmentation features

$$y_i^{(L-1)} = \max\left(0, \sum_{j=1} w_{\mathrm{d},ij}^{(L)} \circledast y_j^{(L)} + c_i^{(L-1)}\right), \tag{5.3}$$

where $y^{(L)} = x^{(L)}$, L denotes the number of layers of the convolutional pathway, $w_{\mathrm{d},ij}^{(L)}$ and $c_i^{(L-1)}$ are trainable parameters of the deconvolutional layer, and $\circledast$ denotes full

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

$$y_i^{(l)} = \max\left(0, \sum_{j=1} w_{\mathrm{d},ij}^{(l+1)} \circledast y_j^{(l+1)} + \sum_{j=1} w_{\mathrm{s},ij}^{(l+1)} \circledast x_j^{(l+1)} + c_i^{(l)}\right), \quad (5.4)$$

where l is the index of a deconvolutional layer with shortcut, and $w_{\mathrm{s},ij}^{(l+1)}$ 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

$$y_1^{(0)} = \mathrm{sigm}\left(\sum_{j=1} \left(w_{\mathrm{d},1j}^{(1)} \circledast y_j^{(1)} + w_{\mathrm{s},1j}^{(1)} \circledast x_j^{(1)}\right) + c_1^{(0)}\right), \quad (5.5)$$

where $\mathrm{sigm}(z)$ denotes the sigmoid function defined as $\mathrm{sigm}(z) = (1 + \exp(-z))^{-1}, z \in \mathbb{R}$. To obtain a binary lesion mask from the probabilistic output of our model, we chose a fixed threshold such that the mean Dice similarity coefficient **?** is maximized on the training set and used the same threshold for the evaluation on the test set.

5.2.3 Gradient Calculation

The parameters of the model can be efficiently learned by minimizing the error E on 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 the sum of squared differences (SSD)

$$E = \frac{1}{2} \sum_{\mathbf{p}} \Big(S(\mathbf{p}) - y^{(2)}(\mathbf{p}) \Big)^2 . \tag{5.6}$$

The partial derivatives of the error with respect to the model parameters can be calculated using the delta rule and are given by

$$\frac{\partial E}{\partial w^{(l)}_{\mathrm{d},ij}} = \delta^{(l-1)}_{\mathrm{d},i} * \tilde{y}^{(l)}_j , \qquad \frac{\partial E}{\partial c^{(l)}_i} = \sum_{\mathbf{p}} \delta^{(l)}_{\mathrm{d},i}(\mathbf{p}), \tag{5.7}$$

$$\frac{\partial E}{\partial w^{(l)}_{\mathrm{s},ij}} = \delta^{(l-1)}_{\mathrm{d},i} * \tilde{x}^{(l)}_j , \tag{5.8}$$

$$\frac{\partial E}{\partial w^{(l)}_{\mathrm{c},ij}} = x^{(l-1)}_i * \tilde{\delta}^{(l)}_{\mathrm{c},j} , \text{ and} \qquad \frac{\partial E}{\partial b^{(l)}_i} = \sum_{\mathbf{p}} \delta^{(l)}_{\mathrm{c},i}(\mathbf{p}). \tag{5.9}$$

For the first layer, $\delta^{(0)}_{\mathrm{d},1}$ can be calculated by

$$\delta^{(0)}_{\mathrm{d},1} = \big(y^{(0)}_1 - S\big) y^{(0)}_1 \big(1 - y^{(0)}_1\big). \tag{5.10}$$

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

$$\delta^{(l)}_{\mathrm{d},j} = \big(\tilde{w}^{(l)}_{\mathrm{d},ij} * \delta^{(l-1)}_{\mathrm{d},i}\big) \mathbb{I}\big(y^{(l)}_j > 0\big), \tag{5.11}$$

$$\delta^{(l)}_{\mathrm{c},i} = \big(w^{(l+1)}_{\mathrm{c},ij} \circledast \delta^{(l+1)}_{\mathrm{c},j}\big) \mathbb{I}\big(x^{(l)}_i > 0\big), \tag{5.12}$$

where l is the index of a deconvolutional or convolutional layer, $\delta^{(L)}_{\mathrm{c},i} = \delta^{(L)}_{\mathrm{d},j}$, and $\mathbb{I}(z)$ denotes the indicator function defined as 1 if the predicate z is true and 0 otherwise. If

a layer connected through a shortcut, $\delta_{c,j}^{(l)}$ needs to be adjusted by propagating the error back through the shortcut connections. In this case, $\delta_{c,j}^{(l)}$ is calculated by

$$\delta_{c,j}^{(l)} = \left(\delta_{c,j}^{(l)\prime} + \tilde{w}_{s,ij}^{(l)} * \delta_{d,i}^{(l-1)}\right)\mathbb{I}\left(x_j^{(l)} > 0\right), \tag{5.13}$$

where $\delta_{c,j}^{(l)\prime}$ 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:

$$E = r\frac{\sum_{\mathbf{p}}\left(S(\mathbf{p}) - y^{(2)}(\mathbf{p})\right)^2 S(\mathbf{p})}{\sum_{\mathbf{p}} S(\mathbf{p})} + (1-r)\frac{\sum_{\mathbf{p}}\left(S(\mathbf{p}) - y^{(2)}(\mathbf{p})\right)^2 \left(1 - S(\mathbf{p})\right)}{\sum_{\mathbf{p}}\left(1 - S(\mathbf{p})\right)}. \tag{5.14}$$

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 preliminarily

experimental results, the algorithm is stable with respect to changes in r, which largely affects the threshold used to binarize the probabilistic output. In all our experiments, a sensitivity ratio between 0.10 and 0.01 yielded very similar results.

To train our model, we must compute the derivatives of the modified objective function with respect to the model parameters. Equations (5.7)–(5.9) and (5.11)–(5.13) are a consequence of the chain rule and independent of the chosen similarity measure. Hence, we only need to derive the update rule for $\delta_{\mathrm{d},1}^{(0)}$. With $\alpha = 2r(\sum_{\mathbf{p}} S(\mathbf{p}))^{-1}$ and $\beta = 2(1-r)(\sum_{\mathbf{p}}(1-S(\mathbf{p})))^{-1}$, we can rewrite E as

$$E = \frac{1}{2}\sum_{\mathbf{p}} \big(\alpha S(\mathbf{p}) + \beta(1-S(\mathbf{p}))\big)\Big(S(\mathbf{p}) - y_1^{(0)}(\mathbf{p})\Big)^2. \tag{5.15}$$

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

$$\delta_{\mathrm{d},1}^{(0)} = \big(\alpha S + \beta(1-S)\big)\big(y_1^{(0)} - S\big)y_1^{(0)}\big(1-y_1^{(0)}\big). \tag{5.16}$$

5.2.4 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 in an early local minimum. Pre-training can be performed layer by layer Hinton

et al. (2006) using a stack of convolutional restricted Boltzmann machines (convRBMs) Lee et al. (2009) (see Fig. Figure 5.1), 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 Fig. Figure 5.1)

$$w_{\mathrm{c}}^{(1)} = \hat{w}^{(1)}, \qquad w_{\mathrm{d}}^{(1)} = 0.5\hat{w}^{(1)}, \qquad w_{\mathrm{s}}^{(1)} = 0.5\hat{w}^{(1)} \tag{5.17}$$

$$b^{(1)} = \hat{b}^{(1)}, \qquad c^{(0)} = \hat{c}^{(1)}, \tag{5.18}$$

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

Recap that finding a learning rate can be challenging. Tried different methods for setting the learning rate. 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.

5.3 Evaluation with the State-of-the-art

To allow for a direct comparison with state-of-the-art lesion segmentation methods, we evaluated our method on the FLAIR, T1-, and T2-weighted MRIs of the 20 publicly available labeled cases from the MS lesion segmentation challenge 2008 Styner et al. (2008), which we downsampled from the original isotropic voxel size of $0.5\,\mathrm{mm}^3$ to an isotropic voxel size of $1.0\,\mathrm{mm}^3$. In addition, we evaluated our method on an in-house

data set from an MS clinical trial of 500 subjects 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 $0.937 \times 0.937 \times 3.000\,\text{mm}$. The main preprocessing steps included rigid intra-subject registration, brain extraction, intensity normalization, and background cropping.

We used a CEN with 32 filters and filter sizes of $9 \times 9 \times 9$ and $9 \times 9 \times 5$ voxels for the challenge and in-house data sets, respectively. 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 trimodal 3D volumes with a resolution of, e.g., $159 \times 201 \times 155$ voxels can be performed in less than one second. As a rough[1] comparison, Ciresan et al. Cireşan et al. (2012) reported that their patch-based method required 10 to 30 minutes to segment a single 2D image with a resolution of 512×512 voxels using four graphics cards, which demonstrates the large speed-ups gained by processing entire volumes.

We evaluated our method on the challenge data set using 5-fold cross-validation and calculated the true positive rate (TPR), positive predictive value (PPV), and Dice similarity coefficient (DSC) between the predicted segmentations and the resampled ground truth. Figure Figure 5.2 shows a comparison of three subjects from the challenge data set. 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

[1] Ciresan et al. used a more complex network that is composed of 11 layers. However, their network was trained on much smaller images, which roughly compensates for the increased complexity.

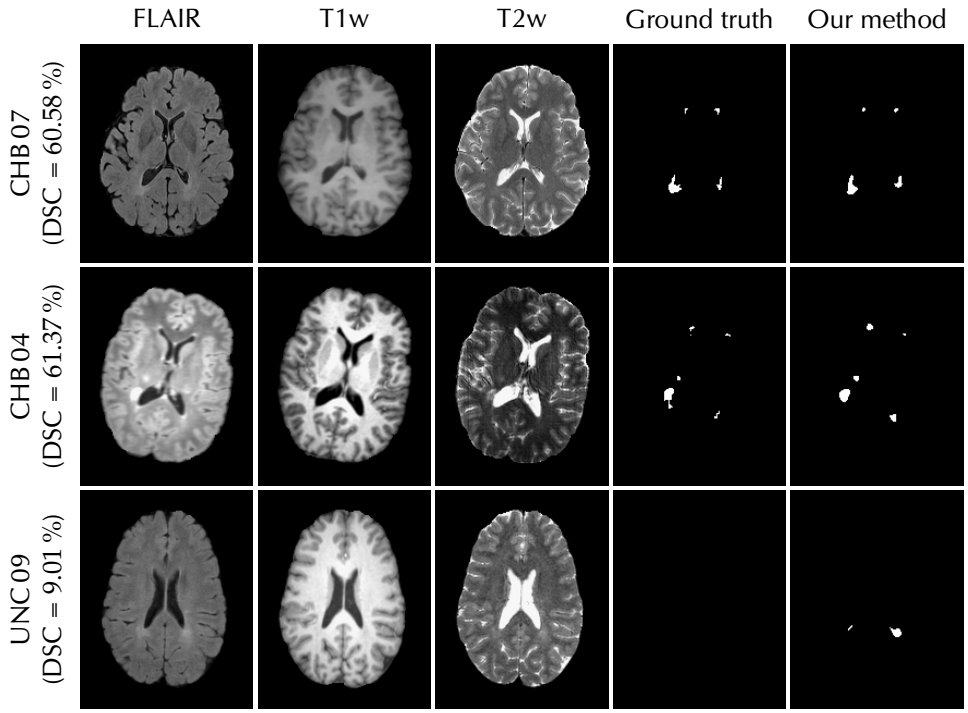


Figure 5.2: Example segmentations of our method for three different subjects from the challenge data set. Our method performed well and consistently despite the large contrast differences seen between the first two rows. In the third row, our method also segmented lesions that have similar contrast, 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.

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 Table 5.1. Our method outperforms the winning method (Souplet et al. Souplet et al. (2008)) of the MS lesion segmentation challenge 2008 and the currently best unsupervised method reported on that data set (Weiss et al. Weiss et al. (2013)) in terms of mean TPR and

Table 5.1: Comparison of our method with state-of-the-art lesion segmentation methods in terms of mean TPR, PPV, and DSC. Our method performs comparably to the best methods reported on the MS lesion segmentation challenge data set.

Method	TPR	PPV	DSC
Souplet et al. Souplet et al. (2008)	20.65	30.00	—
Weiss et al. Weiss et al. (2013)	33.00	36.85	29.05
Geremia et al. Geremia et al. (2010)	39.85	40.35	—
Our method	39.71	41.38	35.52

PPV. Our method performs comparably to a current method (Geremia et al. Geremia et al. (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.

To evaluate the impact of the training set size on the segmentation performance, we trained our model on our in-house data set with a varying number of training samples and calculated the mean DSC on the training and test sets as illustrated in Fig. Figure 5.3. 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 features. At around 100 samples, the model becomes stable in terms of test performance and the small difference 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 %, which shows that the segmentation accuracy can be greatly improved compared to the results on the challenge data set, when a representative training set is available.

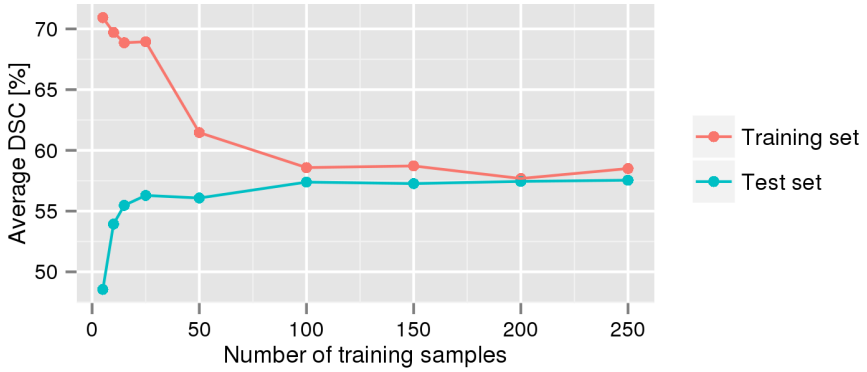


Figure 5.3: Comparison of DSC scores calculated on the training and test sets for varying numbers of training samples. At around 100 samples, the model becomes stable in terms of test performance and the small difference between training and test DSCs, indicating that overfitting of the training data no longer occurs.

5.4 Evaluation on In-house Data

We evaluated the proposed method on a large data set from a multi-center MS clinical trial. The data set, acquired from 67 different scanning sites, consists of 377 pairs of FLAIR and T1-weighted MRIs from 195 subjects with a resolution of $256 \times 256 \times 60$ voxels and a voxel size of $0.936 \times 0.936 \times 3.000\,\text{mm}$. All images were skull-stripped using the brain extraction tool (BET) **?**, followed by an intensity normalization to the interval $[0, 1]$, and a 6 degrees-of-freedom intra-subject registration. To speed-up the training, all images were cropped to a $164 \times 206 \times 52$ voxel region-of-interest with the brain roughly centered. The ground truth segmentations 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., **?** and **?**). To carry out the segmentation, each lesion was manually identified by a trained technician and then interactively grown from the seed point. We used 300 image pairs for pre-training and fine-tuning, and the remaining 77 image pairs for evaluation. Pre-training and fine-tuning were performed using highly optimized GPU-accelerated implementations

of 3D convRBMs and CENs that were developed in-house **?**. Each model was trained using 500 epochs. Pre-training and fine-tuning of a 7-layer CEN-s took approximately 27 hours and 37 hours, respectively, on a single GeForce GTX 780 graphics card. However, once the network is trained, new image pairs can be segmented in less than one second. We compared our results to the lesion masks produced by Lesion-TOADS **?**, a widely used freely available tool for the fully automatic segmentation of MS lesions.

5.4.1 Measures of Segmentation Accuracy

We have used three different measures to evaluate segmentation accuracy. The primary measure of accuracy that we employ is the Dice similarity coefficient (DSC) Dice (1945), which computes a normalized overlap value between the produced and ground truth segmentations, and is defined as

$$\mathrm{DSC} = \frac{2 \times \mathrm{TP}}{2 \times \mathrm{TP} + \mathrm{FP} + \mathrm{FN}}, \tag{5.19}$$

where TP, FP, and FN denote the number of true positives, false positives, and false negatives, respectively. A value of 100 % indicates a perfect overlap of the produced segmentation and the ground truth. The DSC incorporates measures of over- and undersegmentation 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

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{5.20}$$

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

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}'}, \tag{5.21}$$

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.

5.4.2 Comparison of Network Architectures

To determine the effect of network architecture, we compared the segmentation performance of three different networks with each other and with Lesion-TOADS. Specifically, we trained a 3-layer CEN and two 7-layer CENs, one with shortcut connections and one without, on the 300 training pairs. The parameters of the networks are given in Table Table 5.2 and Table Table 5.3. A comparison of the segmentation accuracy of the trained networks and Lesion-TOADS is summarized in Table Table 5.4. All CEN architectures performed significantly better than Lesion-TOADS in overall segmentation accuracy, where the improvements of the mean DSC scores range from 9 pts for the 3-layer CEN to 14 pts for the 7-layer CEN with shortcut connections. The improved segmentation performance is mostly due to a reduction of the false positives. Lesion-TOADS achieved a mean PPV of only 39.86 %, whereas the CEN with shortcut achieved a mean PPV of 66.58 %—an improvement of 27 pts. The mean TPRs were roughly the same (slightly less than 50 %) for all methods except for the 7-layer CEN with shortcut, which performed slightly better than the other methods with a mean TPR of 52.75 %.

Table 5.2: Parameters of the 3-layer CEN used to evaluate different training methods.

Layer type	Kernel Size	#Filters	Image Size
Input	—	—	$164 \times 206 \times 52 \times 2$
Convolutional	$9 \times 9 \times 5 \times 2$	32	$156 \times 198 \times 48 \times 32$
Deconvolutional	$9 \times 9 \times 5 \times 32$	1	$164 \times 206 \times 52 \times 1$

Table 5.3: Parameters of the 7-layer CEN-s used to evaluate different training methods.

Layer type	Kernel Size	#Filters	Image Size
Input	—	—	$164 \times 206 \times 52 \times 2$
Convolutional	$9 \times 9 \times 5 \times 2$	32	$156 \times 198 \times 48 \times 32$
Average Pooling	$2 \times 2 \times 2$	—	$78 \times 99 \times 24 \times 32$
Convolutional	$9 \times 10 \times 5 \times 32$	32	$70 \times 90 \times 20 \times 32$
Deconvolutional	$9 \times 10 \times 5 \times 32$	32	$78 \times 99 \times 24 \times 32$
Unpooling	$2 \times 2 \times 2$	—	$156 \times 198 \times 48 \times 32$
Deconvolutional	$9 \times 9 \times 5 \times 32$	1	$164 \times 206 \times 52 \times 1$

Furthermore, the first experiment showed that increasing the depth of the CEN and adding the shortcut connections improves the segmentation accuracy. Increasing the depth of the CEN from three layers to seven layers improved the mean DSC by 2 pts. The improvement was confirmed to be statistically significant using a one-sided paired t-test (p-value $= 0.002$). Adding a shortcut to the network further improved the segmentation accuracy as measured by the DSC by 3 pts. A second one-sided paired t-test was performed to confirm the statistical significance of the improvement with a p-value of 1.566×10^{-11}.

5.4.3 Comparison for Different Lesion Loads

To examine the effect of lesion load on segmentation performance, we have stratified the test set into five groups based on their reference lesion loads as summarized in Table Table 5.5. Most segmentation performance measures deteriorate with lower lesion

Table 5.4: Comparison of the segmentation accuracy of different CEN models with Lesion-TOADS

Method	TPR [%]	PPV [%]	DSC [%]
3-layer CEN Brosch et al. (2015)	49.62 ± 20.32	59.87 ± 20.95	49.10 ± 15.78
7-layer CEN	49.94 ± 19.96	63.5 ± 20.0	51.04 ± 14.71
7-layer CEN-s	52.75 ± 20.31	66.58 ± 20.71	54.02 ± 15.24
Lesion-TOADS **?**	49.83 ± 14.79	39.86 ± 20.95	40.04 ± 14.86

Note: The table shows the mean and standard deviation of the true positive rate (TPR), positive predictive value (PPV), and Dice similarity coefficient (DSC).

loads, because when there are only a few true lesion voxels, even small segmentation errors can translate into large relative errors. A comparison of segmentation accuracy of a 3-layer CEN and a 7-layer CEN with shortcut for different lesion loads is illustrated in Fig. Figure 5.4. Adding four layers and shortcut connections improves the segmentation accuracy for all lesion load groups, where the improvements are largest for the highest lesion loads. In MS, lesion load is strongly correlated with lesion size, which means that the group with the highest lesion load also contains scans with the largest lesions. 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. 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 than the 3-layer CEN. Consequently, the 7-layer CEN is able to learn a feature set that captures a wider range of lesion sizes, which in turn improves the segmentation accuracy especially for very high lesion loads, where larger lesions are also more prevalent.

Fig. Figure 5.5 shows a comparison of the 7-layer CEN with shortcut and Lesion-TOADS. The CEN approach performs more consistently than Lesion-TOADS for all lesion load groups, but the greatest improvements are for very low to medium lesion loads. For the group with very high lesion loads, Lesion-TOADS achieves a slightly higher mean

Table 5.5: Lesion load classes as used for the detailed analysis.

Group	Lesion load in mm^3	Number of samples
Very low	$[0, 3250]$	17
Low	$(3250, 6500]$	16
Medium	$(6500, 10000]$	18
High	$(10000, 25000]$	18
Very high	> 25000	8

Table 5.6: Comparison of segmentation accuracy for different lesion load categories.

Lesion load	7-layer CEN-s			Lesion-TOADS		
	TPR	PPV	DSC	TPR	PPV	DSC
Very low	50.00	41.15	39.34	49.96	13.09	18.86
Low	61.92	59.01	57.45	52.39	29.95	37.74
Medium	57.64	71.54	61.31	54.17	41.83	46.53
High	51.14	81.11	60.13	47.97	56.56	50.76
Very high	32.82	91.95	48.19	38.88	74.6	50.93

DSC than the CEN approach, but the difference is very small compared to the gains in accuracy achieved by the CEN for the other lesion load groups. Table Table 5.6 shows a more detailed comparison. While the PPV increases consistently with higher lesion loads for both methods, the TPR is highest for low to medium lesion loads and decreases again for high to very high lesion loads. This shows the difficulty for both methods to correctly identify very large lesions that can extend far into the white matter.

5.4.4 Qualitative Results

A qualitative comparison of segmentation performance for four characteristic cases is shown in Fig. Figure 5.6. Our method uses a combination of automatically learned intensity and appearance features, which makes it inherently robust to noise (see Fig. Figure 5.6a),

while still being able to detect small isolated lesions (see Fig. Figure 5.6b). Furthermore, our method is able to learn a wide spectrum of lesion shapes and appearances from training data, which allows our method to correctly identify multiple different types of MS lesions. For example, our method was able to correctly identify the T1 black hole shown in Fig. Figure 5.6c, which present a known limitations of Lesion-TOADS (Shiee et al., 2010) and was partially missed. Figure Figure 5.6d shows one of the most challenging cases for our method. Very large lesions can extend beyond the size of the receptive field of the CEN, which reduces its ability to extract characteristic lesion features. Consequently, in some cases our method can underestimate the size of very large lesions.

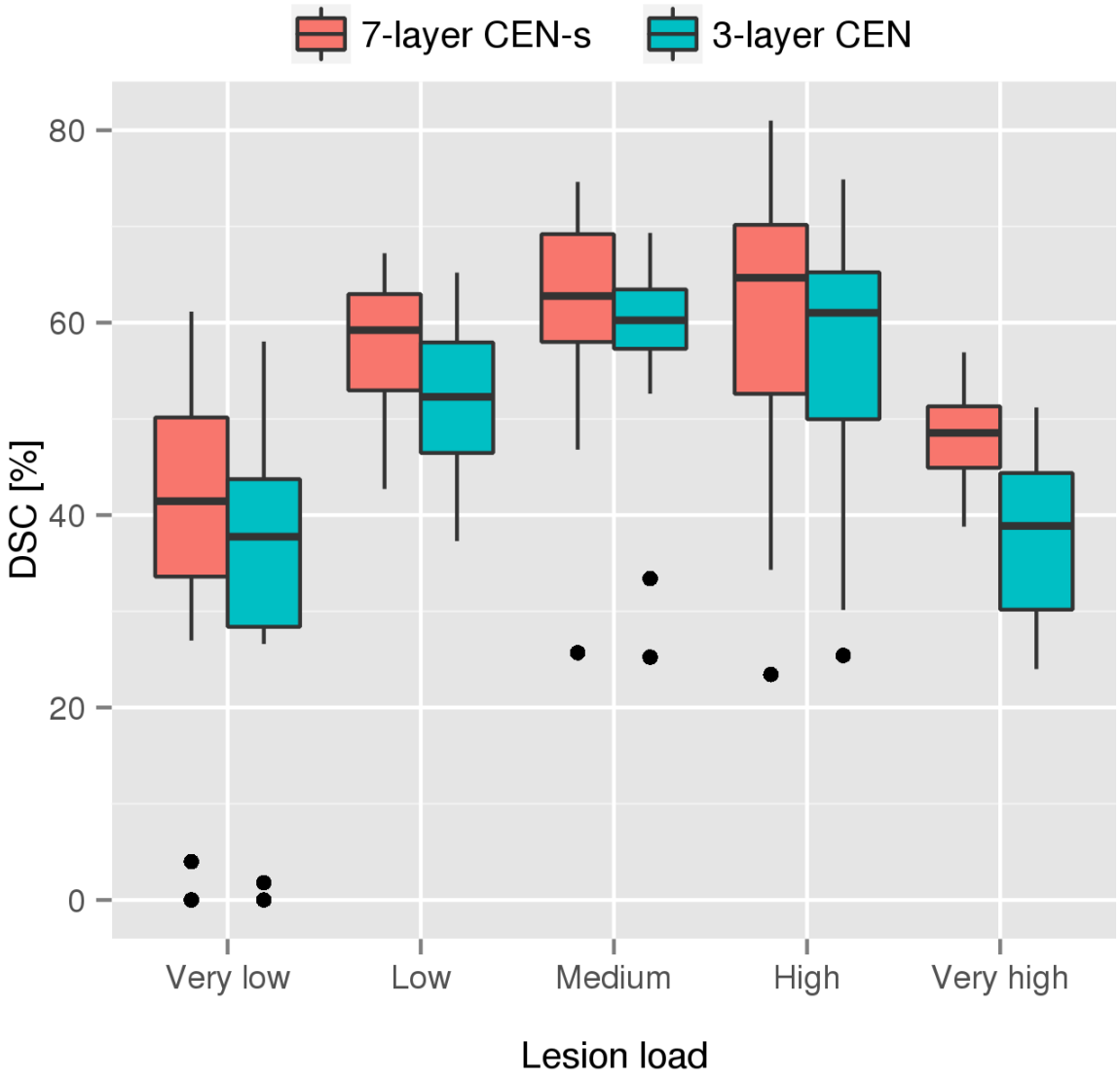


Figure 5.4: Comparison of the segmentation accuracy (DSC) of a 3-layer CEN and a 7-layer CEN-s for different lesion load groups. Adding four layers and shortcut connections improves the performance across all lesion loads, where the improvements are especially large for scans with large lesion loads, which are also correlated with lesion size.

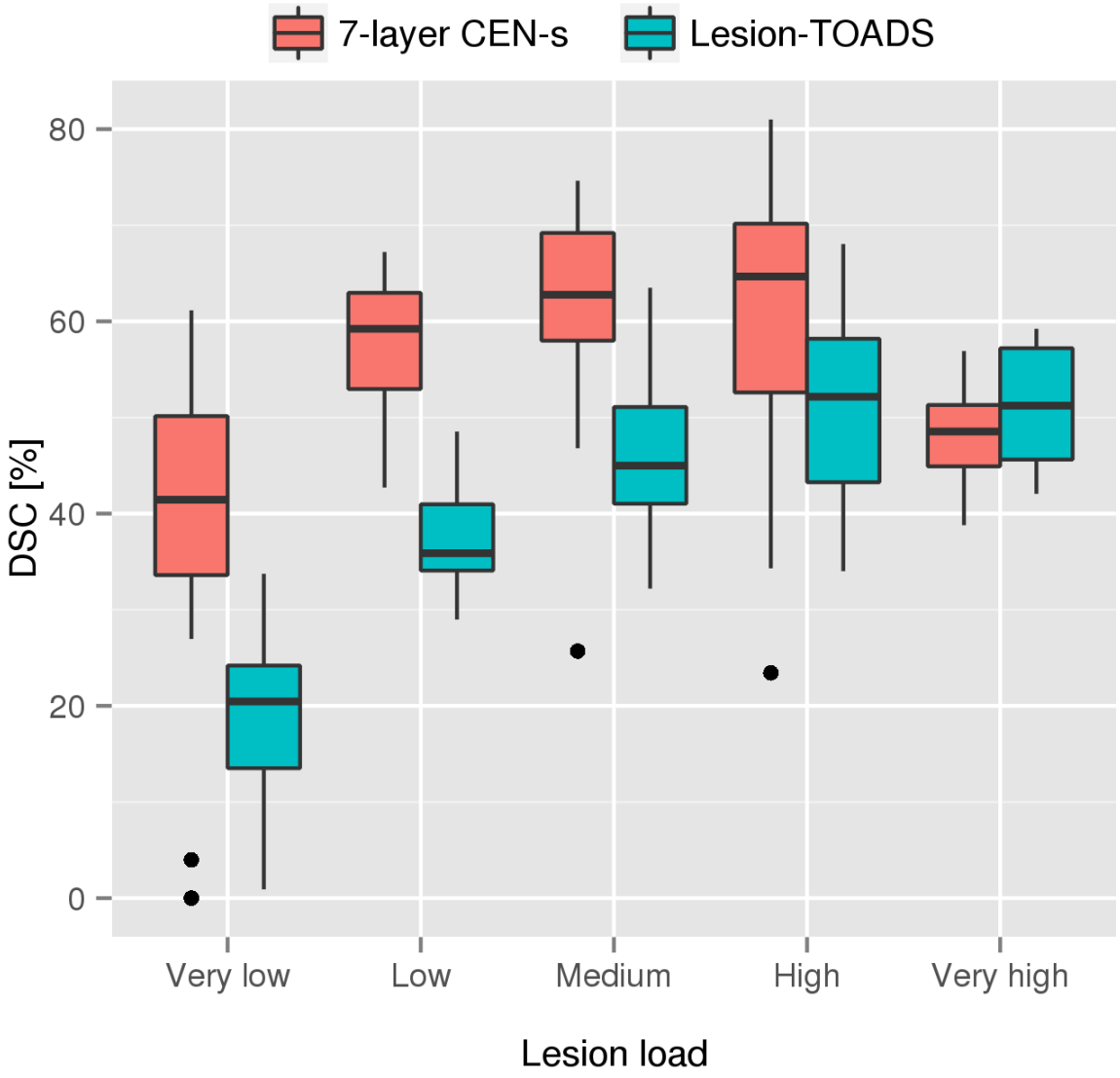
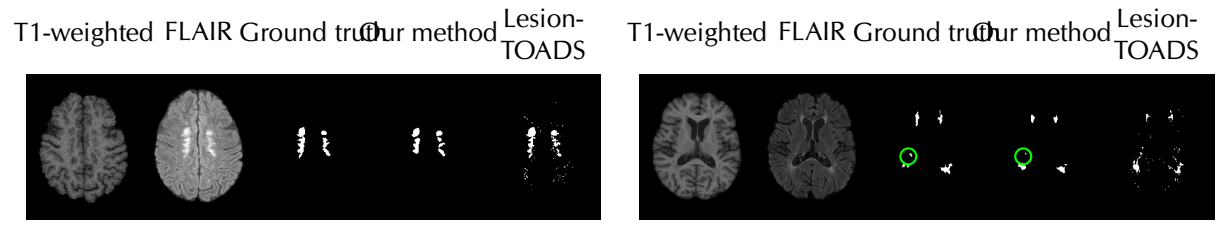


Figure 5.5: Comparison of the segmentation accuracy (DSC) of Lesion-TOADS and a 7-layer CEN with shortcut connections for different lesion loads. The CEN approach is much more sensitive in detecting small lesions, while still being able to detect large lesions.



(a) Robust to noise

(b) Can detect small isolated lesions.

(c) Example of a T1 black hole that was correctly identified by our method.

(d) Very large lesions can be underestimated

Figure 5.6: Four cases illustrating the strengths and limitations of our method compared to Lesion-TOADS. Our method is inherently robust to noise (a), while still being able to detect small isolated lesions (b). Furthermore, our method is able to detect multiple different types of lesions correctly (e.g., T1 black holes). However, in some cases our method can underestimate the size of very large lesions (d).

6 Discussion

What else do I need? General discussion of the role of deep learning for medical image analysis. Maybe.

7 Future Work

Future work: different models like RNNs, LSTMs. Learning invariance to rotations might help especially in 3D. What are other applications? Also need to understand models better. Visualize them maybe? Better training algorithms? More data? Better regularization?

8 Conclusion

Important field and great potential.

Bibliography

Aljabar, Paul, R. Wolz, L. Srinivasan, S. J. Counsell, M. A. Rutherford, A. D. Edwards, J. V. Hajnal and D. Rueckert (2011). *A combined manifold learning analysis of shape and appearance to characterize neonatal brain development.*. IEEE Transactions on Medical Imaging, 30(12):2072–2086. (cited on page 35)

Andersson, Jesper L. R., M. Jenkinson and S. Smith (2007). *Non-linear registration, aka Spatial normalisation*. Technical Report TR07JA2, FMRIB Centre, Oxford, United Kingdom. (cited on page 43)

Belkin, Mikhail and P. Niyogi (2003). *Laplacian eigenmaps for dimensionality reduction and data representation*. Neural Computation, 15(6):1373–1396. (cited on page 33)

Brosch, Tom, Y. Yoo, L. Y. Tang, D. K. Li, A. Traboulsee and R. Tam (2015). *Deep convolutional encoder networks for multiple sclerosis lesion segmentation*. In *A. Frangi et al. (Eds.): MICCAI 2015, Part III, LNCS, vol. 9351*, pp. 3–11. Springer. (cited on pages 53, 54, and 69)

Cayton, Lawrence (2005). *Algorithms for manifold learning*. Technical Report CS2008-0923, University of California, San Diego. (cited on page 35)

Ceccarelli, A., J. Jackson, S. Tauhid, A. Arora, J. Gorky, E. Dell'Oglio, A. Bakshi, T. Chitnis, S. Khoury, H. Weiner and et al. (2012). *The impact of lesion in-painting and registration methods on voxel-based morphometry in detecting regional cerebral gray matter atrophy in multiple sclerosis.*. AJNR American Journal of Neuroradiology, 33(8):1579–1585. (cited on page 34)

Cho, KyungHyun, A. Ilin and T. Raiko (2011). *Improved learning of Gaussian-Bernoulli restricted Boltzmann machines*. In *Artificial Neural Networks and Machine Learning–ICANN 2011*, pp. 10–17. Springer. (cited on page 12)

Cireşan, Dan C., A. Giusti and J. Schmidhuber (2012). *Deep neural networks segment neuronal membranes in electron microscopy images*. In *Advances in Neural Information Processing Systems*, pp. 1–9. (cited on pages 52 and 62)

Dauphin, Yann N., H. de Vries, J. Chung and Y. Bengio (2015). *RMSProp and equilibrated adaptive learning rates for non-convex optimization*. arXiv preprint arXiv:1502.04390. (cited on page 7)

Deng, Jia, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei (2009). *ImageNet: A large-scale hierarchical image database*. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. (cited on page 28)

Dice, Lee R (1945). *Measures of the amount of ecologic association between species*. Ecology, 26(3):297–302. (cited on page 66)

Duchi, John, E. Hazan and Y. Singer (2011). *Adaptive subgradient methods for online learning and stochastic optimization*. The Journal of Machine Learning Research, 12:2121–2159. (cited on page 7)

Etyngier, Patrick, F. Ségonne and R. Keriven (2007). *Shape priors using manifold learning techniques*. In *11th International Conference on Computer Vision*, pp. 1–8. IEEE. (cited on page 34)

Fischer, J S, R. A. Rudick, G. R. Cutter and S. C. Reingold (1999). *The Multiple Sclerosis Functional Composite measure (MSFC): an integrated approach to MS clinical outcome assessment*. Multiple Sclerosis, 5(4):244–250. (cited on page 48)

Fonov, Vladimir, A. C. Evans, K. Botteron, C. R. Almli, R. C. McKinstry and D. L. Collins (2011). *Unbiased average age-appropriate atlases for pediatric studies.*. NeuroImage, 54(1):313–327. (cited on page 42)

Freund, Yoav and D. Haussler (1992). *Unsupervised learning of distributions on binary vectors using two layer networks*. In *Advances in Neural Information Processing Systems*, pp. 912–919. (cited on page 7)

García-Lorenzo, Daniel, S. Francis, S. Narayanan, D. L. Arnold and D. L. Collins (2013). *Review of automatic segmentation methods of multiple sclerosis white matter lesions on conventional magnetic resonance imaging*. Medical image analysis, 17(1):1–18. (cited on page 52)

Gerber, Samuel, T. Tasdizen, P. T. Fletcher, S. Joshi and R. Whitaker (2010). *Manifold modeling for brain population analysis*. Medical Image Analysis, 14(5):643–653. (cited on page 34)

Geremia, Ezequiel, B. H. Menze, O. Clatz, E. Konukoglu, A. Criminisi and N. Ayache (2010). *Spatial decision forests for MS lesion segmentation in multi-channel MR images*. In *Jian, T., Navab, N., Pluim, J., Viergever, M. (eds.) MICCAI 2010, Part I. LNCS, vol. 6362*, pp. 111–118. Springer, Heidelberg. (cited on pages 52 and 64)

Hamm, J., D. Ye and R. Verma (2010). *GRAM: A framework for geodesic registration on anatomical manifolds*. Medical Image Analysis, 14(5):633–642. (cited on page 34)

Hinton, Geoffrey E. (2002). *Training products of experts by minimizing contrastive divergence.*. Neural Computation, 14(8):1771–800. (cited on page 11)

Hinton, Geoffrey E. (2010). *A practical guide to training restricted Boltzmann machines*. Technical Report UTML TR 2010-003, Department of Computer Science, University of Toronto. (cited on page 11)

Hinton, Geoffrey E., S. Osindero and Y.-W. Teh (2006). *A fast learning algorithm for deep belief nets*. Neural Computation, 18(7):1527–1554. (cited on pages 7, 17, 35, 36, 37, 38, 42, 43, 45, and 60)

Hinton, Geoffrey E. and R. Salakhutdinov (2006). *Reducing the dimensionality of data with neural networks*. Science, 313(5786):504–507. (cited on page 60)

Hinton, Geoffrey E. and N. Srivastava (2012). *Improving neural networks by preventing co-adaptation of feature detectors*. arXiv preprint arXiv:1207.0580, pp. 1–18. (cited on page 27)

Hochreiter, Sepp (1991). *Untersuchungen zu dynamischen neuronalen Netzen*. Diploma, Technische Universität München. (cited on page 61)

Hubel, David H. and T. N. Wiesel (1962). *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex*. The Journal of Physiology, 160(1):106. (cited on page 5)

Hubel, David H. and T. N. Wiesel (1968). *Receptive fields and functional architecture of monkey striate cortex*. The Journal of Physiology, 195(1):215–243. (cited on page 5)

Jenkinson, Mark, P. Bannister, M. Brady and S. Smith (2002). *Improved Optimization for the Robust and Accurate Linear Registration and Motion Correction of Brain Images*. NeuroImage, 17(2):825–841. (cited on page 42)

Kang, Kai and X. Wang (2014). *Fully Convolutional Neural Networks for Crowd Segmentation*. arXiv preprint arXiv:1411.4464. (cited on page 53)

Kingma, Diederik and J. Ba (2014). *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980. (cited on page 7)

Krizhevsky, Alex (2012). *High-performance C++/CUDA implementation of convolutional neural networks*. `http://code.google.com/p/cuda-convnet/`. (cited on page 27)

Krizhevsky, Alex, I. Sutskever and G. Hinton (2012). *ImageNet classification with deep convolutional neural networks*. In *Advances in Neural Information Processing Systems*, pp. 1–9. (cited on pages 18 and 24)

LeCun, Yann, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel (1989). *Backpropagation applied to handwritten zip code recognition*. Neural computation, 1(4):541–551. (cited on page 6)

LeCun, Yann, L. Bottou, Y. Bengio and P. Haffner (1998). *Gradient-based learning applied to document recognition*. Proceedings of the IEEE, 86(11):2278–2324. (cited on pages 6, 52, 53, 55, and 58)

Lee, Honglak, R. Grosse, R. Ranganath and A. Y. Ng (2009). *Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations*. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 609–616. ACM. (cited on pages 13, 17, and 61)

Lee, Honglak, R. Grosse, R. Ranganath and A. Y. Ng (2011). *Unsupervised learning of hierarchical representations with convolutional deep belief networks*. Communications of the ACM, 54(10):95–103. (cited on pages 13 and 17)

Marcus, Daniel S, T. H. Wang, J. Parker, J. G. Csernansky, J. C. Morris and R. L. Buckner (2007). *Open Access Series of Imaging Studies (OASIS): cross-sectional MRI data in young, middle aged, nondemented, and demented older adults.*. Journal of Cognitive Neuroscience, 19(9):1498–507. (cited on page 29)

Mathieu, Michael, M. Henaff and Y. LeCun (2014). *Fast Training of Convolutional Networks through FFTs*. In *2nd International Conference on Learning Representations*, pp. 1–9. (cited on pages 18 and 24)

Nair, Vinod and G. E. Hinton (2010). *Rectified linear units improve restricted Boltzmann machines*. In *Proceedings of the 27th Annual International Conference on Machine Learning*, pp. 807–814. (cited on page 12)

Ngiam, Jiquan, A. Khosla, M. Kim, J. Nam, H. Lee and A. Y. Ng (2011). *Multimodal deep learning*. In *28th International Conference on Machine Learning*, pp. 689–696. (cited on page 42)

Petersen, R.C., P. Aisen, L. Beckett, M. Donohue, A. Gamst et al. (2010). *Alzheimer's disease neuroimaging initiative (ADNI)*. Neurology, 74(3):201–209. (cited on page 36)

Polyak, Boris T. and A. B. Juditsky (1992). *Acceleration of stochastic approximation by averaging*. SIAM Journal on Control and Optimization, 30(4):838–855. (cited on page 4)

Raina, Rajat, A. Madhavan and A. Y. Ng (2009). *Large-scale deep unsupervised learning using graphics processors*. In *Proceedings of the 26th annual international conference on machine learning*, pp. 873–880. ACM. (cited on page 17)

Ronneberger, Olaf, P. Fischer and T. Brox (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. In *Proceedings of the 18th International Conferene on Medical Image Computing and Computer Assisted Interventions (MICCAI 2015)*, p. 8. (cited on pages 53 and 54)

Rumelhart, David E., G. E. Hinton and R. J. Williams (1986). *Learning representations by back-propagating errors*. Nature, 323:533–536. (cited on page 4)

Saul, LK and S. Roweis (2003). *Think globally, fit locally: unsupervised learning of low dimensional manifolds*. The Journal of Machine Learning Research, 4:119–155. (cited on page 33)

Scherer, Dominik, A. Müller and S. Behnke (2010). *Evaluation of pooling operations in convolutional architectures for object recognition*. In *International Conference on Artificial Neural Networks*, pp. 92–101. Springer. (cited on pages 15, 27, and 55)

Shiee, Navid, P.-L. Bazin, A. Ozturk, D. S. Reich, P. A. Calabresi and D. L. Pham (2010). *A topology-preserving approach to the segmentation of brain images with multiple sclerosis lesions*. NeuroImage, 49(2):1524–1535. (cited on pages 52 and 71)

Smith, Stephen M., Y. Zhang, M. Jenkinson, J. Chen, P. Matthews, A. Federico and N. De Stefano (2002). *Accurate, Robust, and Automated Longitudinal and Cross-Sectional Brain Change Analysis*. NeuroImage, 17(1):479–489. (cited on page 49)

Souplet, Jean-Christophe, C. Lebrun, N. Ayache and G. Malandain (2008). *An automatic segmentation of T2-FLAIR multiple sclerosis lesions*. In *MIDAS Journal - MICCAI 2008 Workshop*. (cited on pages 52, 63, and 64)

Styner, Martin, J. Lee, B. Chin, M. Chin, O. Commowick, H. Tran, S. Markovic-Plese, V. Jewells and S. Warfield (2008). *3D segmentation in the clinic: A grand challenge II: MS lesion segmentation*. MIDAS Journal - MICCAI 2008 Workshop, pp. 1–6. (cited on page 61)

Sutskever, Ilya, J. Martens, G. Dahl and G. Hinton (2013). *On the importance of initialization and momentum in deep learning*. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pp. 1139–1147. (cited on page 60)

Tenenbaum, J B, V. de Silva and J. C. Langford (2000). *A global geometric framework for nonlinear dimensionality reduction.*. Science (New York, N.Y.), 290(5500):2319–2323. (cited on page 33)

Tieleman, Tijmen (2008). *Training restricted Boltzmann machines using approximations to the likelihood gradient*. In *Proceedings of the 25th International Conference on Machine learning*, pp. 1064–1071. ACM. (cited on page 11)

Vincent, Pascal, H. Larochelle, I. Lajoie, Y. Bengio and P.-A. Manzagol (2010). *Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion*. The Journal of Machine Learning Research, 11:3371–3408. (cited on page 7)

Weiss, Nick, D. Rueckert and A. Rao (2013). *Multiple sclerosis lesion segmentation using dictionary learning and sparse coding*. In *Mori, K., Sakuma, I., Sato, Y., Barillot, C., Navab, N. (eds.) MICCAI 2013, Part I. LNCS 8149*, pp. 735–742. Springer, Heidelberg. (cited on pages 52, 63, and 64)

Werbos, Paul (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University. (cited on page 5)

Wolz, Robin, P. Aljabar, J. V. Hajnal, A. Hammers and D. Rueckert (2010a). *LEAP: learning embeddings for atlas propagation.*. NeuroImage, 49(2):1316–1325. (cited on page 34)

Wolz, Robin, P. Aljabar, J. V. Hajnal, J. Lotjonen and D. Rueckert (2011). *Manifold learning combining imaging with non-imaging information*. In *International Symposium on Biomedical Imaging*, pp. 1637–1640. IEEE. (cited on page 34)

Wolz, Robin, P. Aljabar, J. V. Hajnal, J. Lötjönen and D. Rueckert (2012). *Nonlinear dimensionality reduction combining MR imaging with non-imaging information.*. Medical Image Analysis, 16(4):819–830. (cited on page 35)

Wolz, Robin, P. Aljabar, J. V. Hajnal and D. Rueckert (2010b). *Manifold learning for biomarker discovery in MR imaging*. In Wang, F., P. Yan, K. Suzuki and D. Shen, eds.: *MLMI 2010, LNCS, vol. 6357*, pp. pp. 116–123. Springer. (cited on page 35)

Yoo, Youngjin, T. Brosch, A. Traboulsee, D. K. Li and R. Tam (2014). *Deep Learning of Image Features from Unlabeled Data for Multiple Sclerosis Lesion Segmentation*. In *Wu, G., Zhang D., Zhou L. (eds.) MLMI 2014, LNCS, vol. 8679*, pp. 117–124. Springer, Heidelberg. (cited on page 52)

Zeiler, Matthew D. (2012). *ADADELTA: An adaptive learning rate method*. arXiv preprint arXiv:1212.5701. (cited on page 7)

Zeiler, Matthew D. and R. Fergus (2013). *Stochastic pooling for regularization of deep convolutional neural networks*. In *1st International Conference on Learning Representations*, pp. 1–9. (cited on page 27)

Zeiler, Matthew D., G. W. Taylor and R. Fergus (2011). *Adaptive deconvolutional networks for mid and high level feature learning*. In *2011 IEEE International Conference on Computer Vision*, pp. 2018–2025. Ieee. (cited on pages 53 and 55)