Efficient Training of Convolutional Deep Belief Networks in the Frequency Domain for Application to High-Resolution 2D and...

Article *in* Neural Computation · November 2014

DOI: 10.1162/NECO_a_00682 · Source: PubMed

2 authors:

Tom Brosch
University of British Colu...
17 PUBLICATIONS **126** CITATIONS
SEE PROFILE

Roger Tam
University of British Colu...
71 PUBLICATIONS **469** CITATIONS
SEE PROFILE

LETTER Communicated by Alex Krizhevsky

Efficient Training of Convolutional Deep Belief Networks in the Frequency Domain for Application to High-Resolution 2D and 3D Images

Tom Brosch
brosch.tom@gmail.com
MS/MRI Research Group, Vancouver, BC V6T 2B5, Canada, and Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC V6T 1Z4, Canada

Roger Tam
roger.tam@ubc.ca
MS/MRI Research Group, Vancouver, BC V6T 2B5, Canada, and Department of Radiology, University of British Columbia, Vancouver, BC V5Z 1M9, Canada

Deep learning has traditionally been computationally expensive, and advances in training methods have been the prerequisite for improving its efficiency in order to expand its application to a variety of image classification problems. In this letter, we address the problem of efficient training of convolutional deep belief networks by learning the weights in the frequency domain, which eliminates the time-consuming calculation of convolutions. An essential consideration in the design of the algorithm is to minimize the number of transformations to and from frequency space. We have evaluated the running time improvements using two standard benchmark data sets, showing a speed-up of up to 8 times on 2D images and up to 200 times on 3D volumes. Our training algorithm makes training of convolutional deep belief networks on 3D medical images with a resolution of up to 128 $\times$ 128 $\times$ 128 voxels practical, which opens new directions for using deep learning for medical image analysis.

1 Introduction

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, Osindero, & Teh, 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 $\times$ 28), such as those used for handwritten 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

Neural Computation **27, 211–227** (2015)
doi:10.1162/NECO_a_00682

operations in parallel. Raina and Madhavan (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, Grosse, Ranganath, and Ng (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 of up to 200 × 200 pixels. To speed up the training of convolutional neural networks (CNNs) on high-resolution images, Krizhevsky, Sutskever, and Hinton (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, Henaff, and LeCun (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 recent years, deep learning has shown great potential for medical image analysis. For example, it has been used for segmentation (Ciresan, Giusti, & Schmidhuber, 2012; Liao, Gao, Oto, & Shen, 2013), brain parcellation (Lee, Laine, & Klein, 2011), cancer detection (Cruz-Roa, Edison, Ovalle, Madabhushi, & Gonz, 2013), and deformable registration (Wu, Kim, Wang, & Gao, 2013). However, due to the high computational costs of training large networks, applications have been mostly limited to the analysis of 2D images (Ciresan et al., 2012; Cruz-Roa et al., 2013; Lee, Laine et al., 2011), or small 3D images, frequently extracted as patches of images that are too large to train (Liao et al., 2013; Wu et al., 2013).

Recently we showed the potential of convDBNs for discriminating between Alzheimer's and healthy subjects using their 3D MRI scans (Brosch & Tam, 2013). In this letter, 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 × 128 × 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; rather it 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 run-time 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.

2 Algorithm

In this section, we briefly review the fundamentals of convDBNs (Lee et al., 2009), followed by an introduction to sconvDBNs. Furthermore, we show how an sconvDBN can be mapped to an equivalent convDBN in order to learn the parameters of the sconvDBN using our efficient training algorithms that performs contrastive divergence (CD) (Hinton, 2002) in the frequency domain. (For a comprehensive introduction to DBNs and convDBNs, see Hinton et al., 2006, and Lee, Grosse et al., 2011, respectively.) In this section, for notational simplicity, we assume the input images to be square 2D images, but the model generalizes with little modification to nonsquare images of higher dimensions.

2.1 Convolutional Deep Belief Networks. A convDBN is a multilayer generative model. Each layer of the model can be trained in a greedy layer-wise fashion by treating each pair of adjacent layers as a convolutional restricted Boltzmann machine (convRBM). A convRBM is a probabilistic graphical model composed of layers of visible units ($\mathbf{v}$) and hidden units ($\mathbf{h}$) whose probabilistic relationships can be expressed in terms of convolutions. The joint probability of the visible and hidden units is defined by the energy E of the model as

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \exp(-E(\mathbf{v}, \mathbf{h})), \tag{2.1}$$

where Z is a normalization constant. If the visible and hidden units are binary stochastic units, the energy of a convRBM is defined as

$$\begin{aligned} E(\mathbf{v}, \mathbf{h}) = &- \sum_{i=0}^{N_c-1} \sum_{j=0}^{N_k-1} \sum_{x,y=0}^{N_h-1} \sum_{u,v=0}^{N_w-1} h_{xy}^{j} w_{uv}^{ij} v_{x+u,y+v}^{i} \\ &- \sum_{i=0}^{N_c-1} b_i \sum_{x,y=0}^{N_v-1} v_{xy}^{i} - \sum_{j=0}^{N_k-1} c_j \sum_{x,y=0}^{N_h-1} h_{xy}^{j} \end{aligned} \tag{2.2}$$

Table 1: Key Variables and Notation.

Symbol	Description
w_{uv}^{ij}	Weights of filter kernels connecting visible units v_{xy}^{i} with hidden units h_{xy}^{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
N_{w}^2	Number of weights per filter kernel
$\bullet$	Element-wise product followed by summation
$*_{\mathrm{v}}$	Valid convolution
$\tilde{\mathbf{w}}^{ij}$	Horizontally and vertically flipped version of $\mathbf{w}^{ij}$

Note: For notational simplicity, we assume the input images to be square 2D images.

$$= -\sum_{i=0}^{N_{\mathrm{c}}-1}\sum_{j=0}^{N_{\mathrm{k}}-1}\mathbf{h}^{j}\bullet(\tilde{\mathbf{w}}^{ij}*_{\mathrm{v}}\mathbf{v}^{i}) - \sum_{i=0}^{N_{\mathrm{c}}-1} b_i \sum_{x,y=0}^{N_{\mathrm{v}}-1} v_{xy}^{i}$$
$$- \sum_{j=0}^{N_{\mathrm{k}}-1} c_j \sum_{x,y=0}^{N_{\mathrm{h}}-1} h_{xy}^{j}. \tag{2.3}$$

The key terms and notation are defined in Table 1. At the first layer, the number of channels N_{c} is one for gray-scale images and three for RGB color images. For subsequent layers, N_{c} is equal to the number of filters of the previous layer.

2.2 Strided Convolutional Deep Belief Networks. 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 $N_{\mathrm{h}}^2 = N_{\mathrm{v}}^2/s^2$, 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

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i=0}^{N_{\mathrm{c}}-1}\sum_{j=0}^{N_{\mathrm{k}}-1}\sum_{x,y=0}^{N_{\mathrm{h}}-1}\sum_{u,v=-\lfloor N_{\mathrm{w}}/2\rfloor}^{\lfloor (N_{\mathrm{w}}-1)/2\rfloor} h_{xy}^{j} w_{uv}^{ij} v_{sx+u,sy+v}^{i}$$
$$- \sum_{i=0}^{N_{\mathrm{c}}-1} b_i \sum_{x,y=0}^{N_{\mathrm{v}}-1} v_{xy}^{i} - \sum_{j=0}^{N_{\mathrm{k}}-1} c_j \sum_{x,y=0}^{N_{\mathrm{h}}-1} h_{xy}^{j}. \tag{2.4}$$

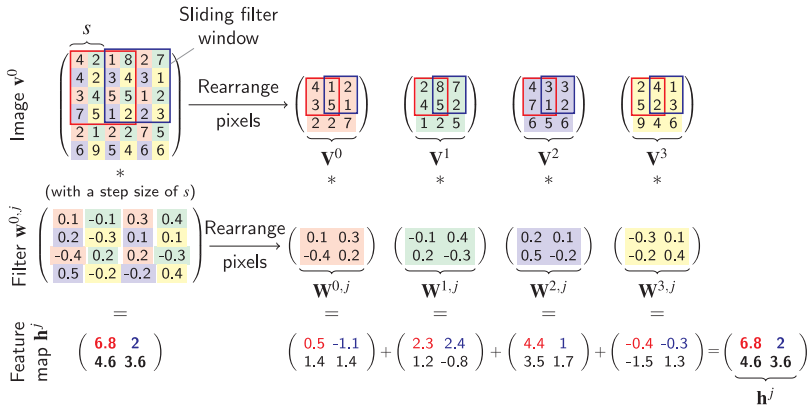


Figure 1: Illustration of convolutions with a sliding window step size $s = 2$ as used during filtering in a sconvDBN. A convolution of a given stride size (left) can be efficiently calculated as the sum of multiple individual convolutions with $s = 1$ (right) after rearranging the pixels of the input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

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 1. This reindexing scheme allows the energy function to be expressed in terms of conventional (stride-1) convolutions, which facilitates training in the frequency domain. The new indices of $V^{i'}_{x'y'}$ and $W^{i'j}_{u'v'}$ can be calculated from the old indices of v^i_{xy} and w^{ij}_{uv} as follows:

$$x' = \lfloor x/s \rfloor \qquad u' = \lfloor u/s \rfloor \tag{2.5}$$

$$y' = \lfloor y/s \rfloor \qquad v' = \lfloor v/s \rfloor \tag{2.6}$$

$$i' = s^2 i + s(y \bmod s) + (x \bmod s). \tag{2.7}$$

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

$$\begin{aligned} E(\mathbf{V}, \mathbf{h}) = &- \sum_{i=0}^{N_C-1} \sum_{j=0}^{N_k-1} \sum_{x,y=0}^{N_h-1} \sum_{u,v=-\lfloor N_W/2 \rfloor}^{\lfloor (N_W-1)/2 \rfloor} h^j_{xy} W^{ij}_{uv} V^i_{x+u,y+v} \\ &- \sum_{i=0}^{N_C-1} b_i \sum_{x,y=0}^{N_V-1} V^i_{xy} - \sum_{j=0}^{N_k-1} c_j \sum_{x,y=0}^{N_h-1} h^j_{xy} \end{aligned} \tag{2.8}$$

$$= - \sum_{i=0}^{N_C-1} \sum_{j=0}^{N_k-1} \mathbf{h}^j \bullet (\tilde{\mathbf{W}}^{ij} * \mathbf{V}^i) - \sum_{i=0}^{N_C-1} b_i \sum_{x,y=0}^{N_V-1} V_{xy}^i - \sum_{j=0}^{N_k-1} c_j \sum_{x,y=0}^{N_h-1} h_{xy}^j, \tag{2.9}$$

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 $N_C = N_c * s^2, N_V^2 = N_v^2/s^2$ and $N_W^2 = N_w^2/s^2$, respectively. The posteriors $p(\mathbf{h} \mid \mathbf{V})$ and $p(\mathbf{V} \mid \mathbf{h})$ can be derived from the energy equation and are given by

$$p\big(h_{xy}^j = 1 \mid \mathbf{V}\big) = \text{sigm}\left(\sum_{i=0}^{N_C-1} (\tilde{\mathbf{W}}^{ij} * \mathbf{V}^i)_{xy} + c_j \right), \tag{2.10}$$

$$p\big(V_{xy}^i = 1 \mid \mathbf{h}\big) = \text{sigm}\left(\sum_{j=0}^{N_k-1} (\mathbf{W}^{ij} * \mathbf{h}^j)_{xy} + b_i \right), \tag{2.11}$$

where $\text{sigm}(x)$ is the sigmoid function defined as $\text{sigm}(x) = (1 + \exp(-x))^{-1}$, $x \in \mathbb{R}$. To train a 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

$$\Delta \mathbf{W}^{ij} \approx \frac{1}{N} \big(\mathbf{V}_n^i * \tilde{\mathbf{h}}_n^j - \mathbf{V}_n'^i * \tilde{\mathbf{h}}_n'^j \big), \tag{2.12}$$

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

$$\mathbb{E}[\mathbf{V}^i \mid \mathbf{h}] = \sum_j \mathbf{W}^{ij} * \mathbf{h}^j + b_i. \tag{2.13}$$

A binary hidden unit can encode only 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 (Nair & Hinton, 2010). The hidden units can be

sampled with

$$\mathbf{h}^j \sim \max(0, \mu^j + \mathcal{N}(0, \operatorname{sigm}(\mu^j))), \tag{2.14}$$

$$\mu^j = \sum_i \tilde{\mathbf{W}}^{ij} * \mathbf{V}^i + c_j. \tag{2.15}$$

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

2.3 Efficient Training of convRBMs 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 the time-reversal property of the Fourier transform and the reality condition $\hat{f}(-\xi) = \overline{\hat{f}(\xi)}$. Using the mappings noted, equations 2.12 to 2.15 can be rewritten as

$$\Delta\hat{\mathbf{W}}^{ij} = \hat{\mathbf{V}}^i \cdot \overline{\hat{\mathbf{h}}^j} - \hat{\mathbf{V}}'^i \cdot \overline{\hat{\mathbf{h}}'^j} \tag{2.16}$$

$$\mathbb{E}[\hat{V}^i_{xy} \mid \hat{\mathbf{h}}] = \begin{cases} \sum_j \hat{W}^{ij}_{xy} \hat{h}^j_{xy} + N_{\mathrm{V}}^2 b_i & \text{for } x, y = 0 \\ \sum_j \hat{W}^{ij}_{xy} \hat{h}^j_{xy} & \text{for } x, y \neq 0 \end{cases} \tag{2.17}$$

$$\hat{\mathbf{h}}^j \sim \mathscr{F}(\max(0, \mathscr{F}^{-1}(\hat{\mu}^j) + c_j + \mathcal{N}(0, \sigma^2))) \tag{2.18}$$

$$\hat{\mu}^j = \sum_i \overline{\hat{\mathbf{W}}^{ij}} \cdot \hat{\mathbf{V}}^i \tag{2.19}$$

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

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

$\Delta\mathbf{W} = 0$
foreach image $\mathbf{V} \in \mathscr{D}$ **do**
  $\mathbf{V}' = 0$
  for $j = 0$ **to** $N_\mathrm{k} - 1$ **do**
    $\mu^j = \sum_i \tilde{\mathbf{W}}^{ij} * \mathbf{V}^i + c_j$
    $h^j \sim \max(0, \mu^j + \mathscr{N}(0, \mathrm{sigm}(\mu^j)))$
    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**
    $\mu'^j = \sum_i \tilde{\mathbf{W}}^{ij} * \mathbf{V}'^i + c_j$
    $\mathbf{h}'^j \sim \max(0, \mu'^j + \mathscr{N}(0, \mathrm{sigm}(\mu^j)))$
    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 $\hat{\mathscr{D}} = \{\hat{\mathbf{V}}_n \mid n \in [0, N-1]\}$
output: Weights gradient $\Delta\hat{\mathbf{W}}$

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

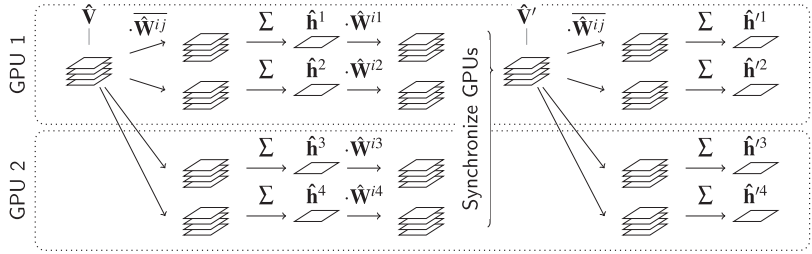


Figure 3: Distribution of operations and memory for training a convRBM using multiple GPUs. All filter operations assigned to each GPU are performed independent of other GPUs. Synchronization of GPUs is required only to accumulate the reconstructed visible units $\hat{\mathbf{V}}'$.

the filter kernels also does not add significantly to the total running time of the training algorithm.

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

If the size of an sconvDBN exceeds the available amount of memory of a single GPU, training can be distributed by assigning different channels to separate GPUs as illustrated in Figure 3. The inference of the hidden units $\hat{\mathbf{h}}^j$ of different channels and the estimation of filter increments $\Delta\hat{\mathbf{W}}^{ij}$ of different filters are independent of each other. Therefore, the processing and storing of filters, filter increments, and hidden units can be distributed over multiple GPUs without the need to transfer memory content between GPUs, which would decrease the GPU utilization. To infer the reconstructed visible units $\hat{\mathbf{V}}'$, each GPU calculates a separate reconstruction from a distinct set of filters in parallel. Afterward, the separate reconstructions are added up sequentially to create the complete reconstruction, which requires the synchronization of GPUs and consequently decreases the GPU utilization.

However, this operation is relatively fast compared to the large number of filtering operations and therefore does not add significantly to the total running time.

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 be done only 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:

$$4N_{\mathrm{v}}^2 N_{\mathrm{c}} + \frac{4N_{\mathrm{v}}^2 N_{\mathrm{k}}}{s^2} + 4N_{\mathrm{v}}^2 N_{\mathrm{k}} N_{\mathrm{c}}. \tag{2.20}$$

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

$$4N_{\mathrm{v}}^2 N_{\mathrm{b}} N_{\mathrm{c}} + \frac{4N_{\mathrm{v}}^2 N_{\mathrm{b}} N_{\mathrm{k}}}{s^2} + 4N_{\mathrm{w}}^2 N_{\mathrm{k}} N_{\mathrm{c}}, \tag{2.21}$$

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

$$4N_{\mathrm{v}}^2 N_{\mathrm{b}} N_{\mathrm{c}} + 4N_{\mathrm{v}}^2 N_{\mathrm{b}} N_{\mathrm{k}} + 4N_{\mathrm{v}}^2 N_{\mathrm{k}} N_{\mathrm{c}}. \tag{2.22}$$

Table 2: Comparison of the Memory Required for Storing Key Variables Using Different Training Methods: Our Method (Freq), Krizhevsky et al.'s (2012) Spatial Domain Method (Spat), and Mathieu et al.'s (2014) Method Using Batched FFTs (B-FFT).

							Memory in MB		
Layer	N_b	N_v	N_c	N_w	N_k	s	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

Notes: A comparison with Mathieu et al.'s method could not be made for the first layer because that method does not support strided convolutions. In all layers, our method consumes less memory for storing the key variables than the other two methods.

Table 2 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. (2014) 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 it still outperforms batched learning in the spatial domain in terms of speed (see section 3), despite not using batches.

3 Evaluation

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 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 (Hinton & Srivastava, 2012; Scherer, Müller, & Behnke, 2010; Zeiler & Fergus, 2013) that calculates the convolution of batches of 2D images and 2D filter kernels in parallel (note that this method cannot be applied to 3D images, so it was only used for the 2D experiments), and (3) our implementation that calculates convolutions using FFTs but without mapping the other operations that would allow the algorithm to stay in the frequency domain when not computing convolutions. The parameters that we used for training convRBMs on 2D and 3D images are summarized in Table 3. The key parameters that we

Table 3: Training Parameters.

	ImageNet (2D)		OASIS (3D)	
Parameter	First Layer	Second Layer	First Layer	Second 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

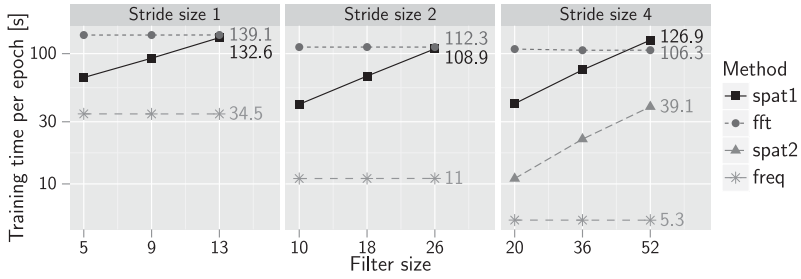
Table 4: Hardware Specifications of Our Test System.

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

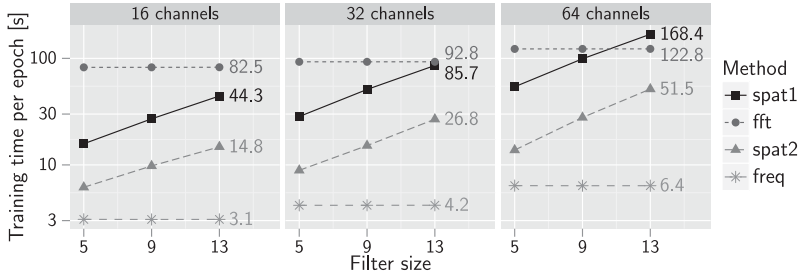
varied for our experiments are the filter size and stride size of the first layer and the filter size and the number of channels of the second layer. Because the number of channels of the second layer is equal to the number of filters of the first layer, we also varied the number of filters of the first 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 letter. The hardware details of our test environment are summarized in Table 4.

For the comparison on 2D images, we used a data set of 256 natural color images from the ImageNet data set (Deng et al., 2009). All images were resampled to a resolution of 512 $\times$ 512 pixels per color channel. For the evaluation on 3D images, we used 100 magnetic resonance images (MRIs) of the brain from the OASIS data set (Marcus et al., 2007). We resampled all volumes to a resolution of 128 $\times$ 128 $\times$ 128 voxels and a voxel size of 2 mm $\times$ 2 mm $\times$ 2 mm .

3.1 Running Time Analysis on 2D Color Images (ImageNet). Figure 4a shows a comparison of running times for training the first sconvRBM layer on 256 images with varying filter and stride sizes. Due to internal limitations of Krizhevsky's convolution implementation, it cannot be applied to images with a resolution of 512 $\times$ 512 pixels when using a stride size smaller than four, and those comparisons could not be made. Our frequency domain



(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 4: 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.

implementation is 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 one, the impact of the convolution implementation on the total running time is relatively low because the computational bottleneck is the inference and sampling of the hidden units. As the number of hidden units decreases with larger strides, the running time becomes more dependent on the time spent to calculate convolutions. Hence, the differences between the four methods are more pronounced for larger strides. For a stride of four, training in the frequency domain is 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

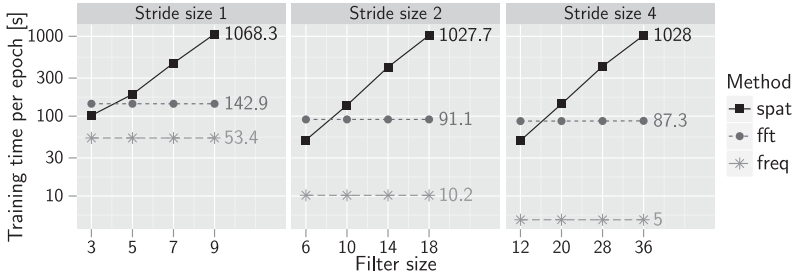
is the slowest method for all stride sizes and 2D filter sizes up to 44, largely due to the cost of calculating Fourier transforms.

Figure 4b shows a similar comparison for training the second convRBM layer for a stride size of one and varying filter sizes and numbers of channels. In contrast to training the first layer, training times mostly depend on the calculation of convolutions, where the impact of calculating convolutions on the total running time increases with the increasing number of channels. Training in the frequency domain is 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 nonbatched 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, we show that the proposed method yields a speed-up of 7 to 8 times compared to state-of-the-art GPU implementations.

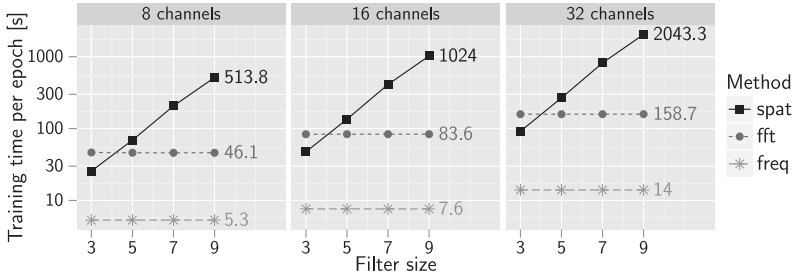
3.2 Running Time Analysis on 3D Volumes (OASIS). Figure 5 shows the comparison of running times for training a first- and second-layer sconvRBM on 3D volumes for varying filter sizes, stride sizes, and varying numbers of channels. In contrast to training on 2D images, the computational costs of calculating 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 five and larger, our frequency domain implementation is 3.5 to 200 times faster than our spatial domain implementation using single-image convolutions and 2.7 to 17 times faster than calculating convolutions by FFTs. Similar to the results on 2D images, training times of the first layer using a stride of one depend strongly on the time required to calculate the expectation of the hidden units and 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.

4 Conclusion

We have presented a fast training method for convolutional models that performs training in the frequency domain in order to replace the time-consuming computation of convolutions with simple element-wise multiplications. We have shown that it is also essential to map the other



(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 5: 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).

operations to frequency space wherever possible to minimize the number of Fourier transforms and that this greatly decreases training times over performing only the convolutions in the frequency domain. In addition, our method can be efficiently implemented on the GPU and is faster than a highly optimized GPU implementation of batched convolutions in the spatial domain. We have evaluated the running time improvements using two standard benchmark data sets, showing a speed-up of up to 8 times on 2D images from the ImageNet dataset and up to 200 times on 3D volumes from the OASIS data set.

Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Milan and Maureen Ilich Foundation. In addition, data used in the evaluation of this work were funded by the

Open Access Structural Imaging Series (grants P50 AG05681, P01 AG03991, R01 AG021910, P20 MH071616, U24 RR021382). We thank the anonymous reviewers for many helpful comments that improved the letter.

References

Brosch, T., & Tam, R. (2013). Manifold learning of brain MRIs by deep learning. In K. Mori, I. Sakuma, Y. Sato, C. Barillot, & N. Navab (Eds.), *Medical image computing and computer-assisted intervention* 2013, Part II, LNCS 8150 (pp. 633–640). Berlin: Springer.

Ciresan, D., Giusti, A., & Schmidhuber, J. (2012). Deep neural networks segment neuronal membranes in electron microscopy images. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems, 25* (pp. 1–9). Red Hook, NY: Curran.

Cruz-Roa, A. A., Edison, J., Ovalle, A., Madabhushi, A., & Gonz, F. A. (2013). A deep learning architecture for image representation, visual interpretability and automated basal-cell carcinoma cancer detection. In K. Mori, I. Sakuma, Y. Sato, C. Barillot, & N. Navab (Eds.), *Medical image computing and computer-assisted intervention*, 2013, Part II, LNCS 8150 (pp. 403–410). Berlin: Springer.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 248–255). Piscataway, NJ: IEEE.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, *14*(8), 1771–1800.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*(7), 1527–1554.

Hinton, G. E., & Srivastava, N. (2012). *Improving neural networks by preventing co-adaptation of feature detectors*. arXiv preprint arXiv:1207.0580.

Krizhevsky, A. (2012). *High-performance C++/CUDA implementation of convolutional neural networks*. http://code.google.com/p/cuda-convnet/

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Weinberger (Eds.), *Advances in neural information processing systems, 25* (pp. 1–9). Red Hook, NY: Curran.

Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (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). New York: ACM.

Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2011). Unsupervised learning of hierarchical representations with convolutional deep belief networks. *Communications of the ACM*, *54*(10), 95–103.

Lee, N., Laine, A., & Klein, A. (2011). Towards a deep learning approach to brain parcellation. In *Proceedings of the IEEE International Symposium on Biomedical Imaging* (pp. 321–324). Piscataway, NJ: IEEE.

Liao, S., Gao, Y., Oto, A., & Shen, D. (2013). Representation learning: A unified deep learning framework for automatic prostate MR segmentation. In K. Morin,

I. Sakuma, Y. Sato, C. Barillot, & N. Navab (Eds.), *Medical image computing and computer-assisted intervention*, 2013, Part II, LNCS 8150 (pp. 254–261). Berlin: Springer.

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

Mathieu, M., Henaff, M., & LeCun, Y. (2014). Fast training of convolutional networks through FFTs. In *Proceedings of the 2nd International Conference on Learning Representations* (pp. 1–9). arXiv.org

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th Annual International Conference on Machine Learning* (pp. 807–814). Madison, WI: Omnipress.

Raina, R., & Madhavan, A. (2009). Large-scale deep unsupervised learning using graphics processors. In L. Bottou & M. Littman (Eds.), *Proceedings of the 26th International Conference on Machine Learning*. Madison, WI: Omnipress.

Scherer, D., Müller, A., & Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In *Proceedings of the 20th International Conference on Artificial Neural Networks* (pp. 92–101). Berlin: Springer.

Wu, G., Kim, M., Wang, Q., & Gao, Y. (2013). Unsupervised deep feature learning for deformable registration of MR brain images. In K. Mori, I. Sakuma, Y. Sato, C. Barillot, & N. Navab (Eds.), *Medical image computing and computer-assisted intervention*, 2013, Part II, LNCS 8150 (pp. 649–656).

Zeiler, M. D., & Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. In *Proceedings of the 1st International Conference on Learning Representations* (pp. 1–9). arXiv.org

Received March 27, 2014; accepted July 12, 2014.