THE UNIVERSITY *of* EDINBURGH

Statistical Models for Natural Scene Data

Jyri J. Kivinen

Doctor of Philosophy

Institute for Adaptive and Neural Computation
School of Informatics
University of Edinburgh

2013

Abstract

This thesis considers statistical modelling of natural image data. Obtaining advances in this field can have significant impact for both engineering applications, and for the understanding of the human visual system. Several recent advances in natural image modelling have been obtained with the use of unsupervised feature learning. We consider a class of such models, restricted Boltzmann machines (RBMs), used in many recent state-of-the-art image models. We develop extensions of these stochastic artificial neural networks, and use them as a basis for building more effective image models, and tools for computational vision.

We first develop a novel framework for obtaining Boltzmann machines, in which the hidden unit activations co-transform with transformed input stimuli in a stable and predictable way throughout the network. We define such models to be *transformation equivariant*. Such properties have been shown useful for computer vision systems, and have been motivational for example in the development of steerable filters, a widely used classical feature extraction technique. Translation equivariant feature sharing has been the standard method for scaling image models beyond patch-sized data to large images. In our framework we extend shallow and deep models to account for other kinds of transformations as well, focusing on in-plane rotations.

Motivated by the unsatisfactory results of current generative natural image models, we take a step back, and evaluate whether they are able to model a subclass of the data, natural image textures. This is a necessary subcomponent of any credible model for visual scenes. We assess the performance of a state-of-the-art model of natural images for texture generation, using a dataset and evaluation techniques from in prior work. We also perform a dissection of the model architecture, uncovering the properties important for good performance. Building on this, we develop structured extensions for more complicated data comprised of textures from multiple classes, using the single-texture model architecture as a basis. These models are shown to be able to produce state-of-the-art texture synthesis results quantitatively, and are also effective qualitatively. It is demonstrated empirically that the developed multiple-texture framework provides a means to generate images of differently textured regions, more generic globally varying textures, and can also be used for texture interpolation, where the approach is radically different from the others in the area.

Finally we consider visual boundary prediction from natural images. The work aims to improve understanding of Boltzmann machines in the generation of image segment boundaries, and to investigate deep neural network architectures for learning the boundary detection problem. The developed networks (which avoid several hand-crafted model and feature designs commonly used for the problem), produce the fastest reported inference times in the literature, combined with state-of-the-art performance.

Acknowledgements

It has been an honour to be able to work with my primary supervisor, Prof. Chris Williams. His brilliance, expert knowledge, grandmaster-like supervision skills, and friendliness and willingness to discuss and comment things from the high-level down to the very details have helped and motivated my research and writing up significantly. He has also made it possible for me to attend excellent summer schools and research meetings in several countries, and to meet many interesting people also outside the events.

I have received supervision also from Dr. Iain Murray and Prof. Stephen McKenna, for which I'm very grateful. Dr. Nicolas Heess has been involved in the research related to Chapter 5 work via discussions which have been useful for the research and the writing up. His friendliness and the relaxed-yet-effective working atmosphere he creates have made the discussions a pleasure and motivational.

I would like to thank the Probabilistic Inference Group (PIGS) in Edinburgh which I have been part of. This elite group of machine learning researchers provided me a stimulating working atmosphere, constant opportunities to hear about top-of-the-line developments, and also comments about my work.

I'm very grateful to several sources for funding: The Scottish Informatics and Computer Science Alliance (SICSA), Engineering and Physical Sciences Research Council (EPSRC) and the Informatics Graduate School provided funding for my thesis work. The PASCAL Network of Excellence has provided money for conference travels, and also for summer school activities I'm happy to have been able to participate in. The Canadian Institute for Advanced Research (CIfAR) provided funding for my participation to their excellent Neural Computation and Adaptive Perception (NCAP) Summer school hosted by the Machine Learning Group at the Department of Computer Science, University of Toronto. The Rank Prize Funds paid for my expenses to their excellent Symposium on Machine Learning and Computer Vision in Grasmere, UK.

Professors Pirkko Oittinen, Erkki Oja, Michael I. Jordan, and Erik B. Sudderth have advanced my career starting before Edinburgh and have contributed to this opportunity I was given, much thanks also again to Prof. Chris Williams.

Last but not least, I would like to thank my friends and colleagues (esp. Peter Orchard) for their support, and especially the enormous support from my parents.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Jyri J. Kivinen*)

Table of Contents

Chapter 1

Introduction

1.1 Statistical Modelling of Natural Image Data

The main theme in the thesis is building better statistical models for natural image data, photographic images of the human living environment. The human visual system has evolved to process the underlying high-dimensional and highly-variable stimuli in an efficient manner, to a degree which is fully subconscious for most humans. For computational vision, many obstacles still remain. Obtaining advances in the field can have significant impact for several engineering applications (both existing and novel), and for the understanding of the human visual system and also the brain [Hyvarinen et al., 2009]

Early computational vision was highly rule-based and hand-engineered, but statistical learning-based methods have become increasingly popular in modern approaches. The amount of human supervision in the model design has also been decreasing, with unsupervised feature learning becoming increasingly common, replacing hand-engineered feature designs. The workhorse in many recent state-of-the-art image models is a stochastic artificial neural network, the restricted Boltzmann machine. This class of models will be the main methodological basis for the work developed here.

There are several justifications for studying this class of models, including theoretical ones such as maximum-entropy connections of the models, and empirical ones such as the similarity of the features learned by these architectures with those found via brain-imaging methods. In several recent application domains, including computer vision applications, hand-engineered features and ad-hoc model designs have been 'losing the contests' to these principled statistical

machine learning methods. The ability to simulate from the model is one of the motivations to study generative models, as the researcher is able to see what the model 'believes in', giving possible leverage for hypotheses on model improvement.

1.2 Building Structure into Statistical Models of Images

Boltzmann machines have shown promise for several statistical modelling tasks [Bengio et al., 2013]. However, they have been unsatisfactory in synthesizing realistic looking natural images, similar to other generative models for natural images. Additionally comparisons involving undirected models are problematic under the assumption of generic natural images. This is due to quantitative comparison being computationally demanding, as classical methods such as exact likelihood computations are intractable. Also, it is hard to qualitatively compare the approaches under the highly variant stimuli by a human observer.

Motivated by these facts, the thesis builds on earlier generative natural image and texture modelling work, considering a structured approach, starting from simple to more complex with respect to the data and the model properties. The thesis considers and develops methods for low-level and mid-level natural image modelling as described in more detail in the following section, and also provides suggestions for future work on higher level image understanding.

1.3 Thesis Structure

The following chapter contains the background material. In Chapter 3 we consider unsupervised feature learning from natural images using Boltzmann machines. We have developed a novel framework for obtaining Boltzmann-machines and their deep extensions, in which the hidden unit activations co-transform with transformed input stimuli in a stable and predictable way throughout the network. We define such models to be *transformation equivariant*. Such property is clearly useful for computer vision systems, and have been also motivational for example in the development of steerable filters [Simoncelli et al., 1992]. Translation equivariant feature sharing (also known as convolutional feature sharing) has

been the method for scaling image models (including those based on stochastic neural networks) beyond patch-sized data onto large images. In our framework we extend shallow and deep models to account for other kinds of transformations as well, focusing on in-plane rotations in the thesis, and in **Kivinen and Williams [2011]**.

Motivated by the unsatisfactory natural image generative modelling results by the current methods, the thesis takes a step back, and evaluates whether they are able to model a subclass of the data, natural image textures, a necessary subcomponent of any credible model for visual scenes. In Chapter 4, we first assess the generative performance of a state-of-the-art model on natural images in texture generation, and perform a dissection of the model architectures, to develop understanding of the properties important for modelling the data. Our models produce state-of-the-art results quantitatively and are also qualitatively effective, and realistic. We then develop in the chapter and in **Kivinen and Williams [2012]** structured extensions for more complicated data, textures from multiple classes, using the single-texture model architecture as a basis. In the chapter it is demonstrated empirically that the multiple-texture framework provides a means to generate images of differently textured regions, and also more generic globally varying textures, and can be used for texture interpolation.

Chapter 5 considers visual boundary prediction from natural images. The work aims to improve understanding of Boltzmann machines in the generation of image segment boundaries, and to investigate deep neural network architectures for learning the boundary detection problem. The developed networks, which avoid several hand-engineed model and feature designs commonly used for the problem, produce state-of-the-art prediction results with fastest reported inference times in the literature at the prediction performance level. A manuscript [**Kivinen et al., 2013b**] is in preparation on the work.

Chapter 6 summarizes the work in the thesis, and provides a discussion on future work.

Chapter 2

Background

This chapter provides background for the techniques developed in the thesis, and discussion of closely related techniques. The main focus will be describing undirected graphical models, restricted Boltzmann machines in particular, and learning methods for them.

We start in Section 2.1 by describing undirected graphical models. Section 2.2 then discusses acyclic directed graphical models. Finally, Section 2.3 considers learning and inference methods for stochastic and deterministic neural networks.

2.1 Undirected Graphical Models

Undirected graphical models, also called as Markov random fields (MRFs), are probabilistic models on a set of random variables. They can be visualized graphically with networks consisting of nodes/units for the random variables, and undirected connections or links between the nodes, according to the probabilistic dependencies the model is set to define. Figure 2.1 visualizes an example MRF. A set of nodes which are fully connected to each other are called cliques, and maximal cliques are cliques that cannot be expanded to having additional nodes (without losing the property of being fully connected and thus being a clique).

The joint probability density of the J random variables $\{x_1, \ldots, x_J\}$ in the network can be written as a normalized product of clique-specific terms (see for example Koller and Friedman [2009]):

$$p(x_1, \ldots, x_J) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \Phi_c(\mathbf{x}_c), \tag{2.1}$$

where Z is a normalization-constant, also called the partition function, $\mathbf{x}_c$ denotes

nodes within a clique c, each of which are from the set of cliques $\mathcal{C}$ of the graph, and have an associated strictly positive potential function Φ_c acting on the joint values of the clique nodes. The cliques can be the maximal cliques of the graph, but do not need to be.

The networks contain observed/visible units, and can also have hidden/latent ones. The thesis uses a common graphical notation of shading the observed variables. The theory allows partially observed networks to be converted onto fully observed ones, by the so-called process of marginalization. It is mathematically equivalent to summing or integrating the joint probability density of the visible and hidden units over the set of hidden unit configurations. Graphically, when the process is performed on a single node, all nodes connected to the node being integrated out become coupled, and connections will be added between them, if not already existing. The same processes are applied, or are conformed to, when marginal densities of variables (of any sets of variables) are computed.

Let us assume a partition of the nodes into three sets, $\mathbf{x}_{\mathcal{S}_1}$, $\mathbf{x}_{\mathcal{S}_2}$, and $\mathbf{x}_{\mathcal{S}_3}$. The global Markov property states that nodes $\mathbf{x}_{\mathcal{S}_1}$ will be independent from $\mathbf{x}_{\mathcal{S}_2}$, conditional on $\mathbf{x}_{\mathcal{S}_3}$, if the former two sets are separated in the graph by $\mathbf{x}_{\mathcal{S}_3}$. Assume that nodes $\mathbf{x}_{\mathcal{S}_3}$ are the neighbors of a node x_j (in other words $\mathbf{x}_{\mathcal{N}_j}$). This obviously then implies that conditional on the neighboring variables $\mathbf{x}_{\mathcal{N}_j}$, x_j will be independent on the rest of the graph $\mathbf{x}_{\setminus j,N_j}$. Mathematically expressed,

$$p\big(x_j \mid \mathbf{x}_{\setminus j}\big) = p\Big(x_j \mid \mathbf{x}_{\mathcal{N}_j}, \mathbf{x}_{\setminus j,N_j}\Big) = p\big(x_j \mid \mathbf{x}_{\mathcal{N}_j}\big)\,, \tag{2.2}$$

where the backslash-operator denotes an exclusion operation, and for example $\mathbf{x}_{\setminus j}$ denotes the set $\mathbf{x}$ excluding x_j. The Hammersley-Clifford Theorem (see for example Koller and Friedman [2009]) shows further that the functional form of the joint density follows that of equation 2.1.

It is then easy to see that a simple local connectivity structure can be very beneficial in terms of inference and learning, and that marginalization might not be computationally feasible in practise. Innovating (additional) hidden units onto the model on the other hand can be a way of making local connectivity structures simpler, yet maintaining the flexibility of the model. Both of these two strategies are heavily used in graphical model and algorithm design.

Energy-based statistical models define the joint probability density of the random variables via an energy-function. The probability is defined using the

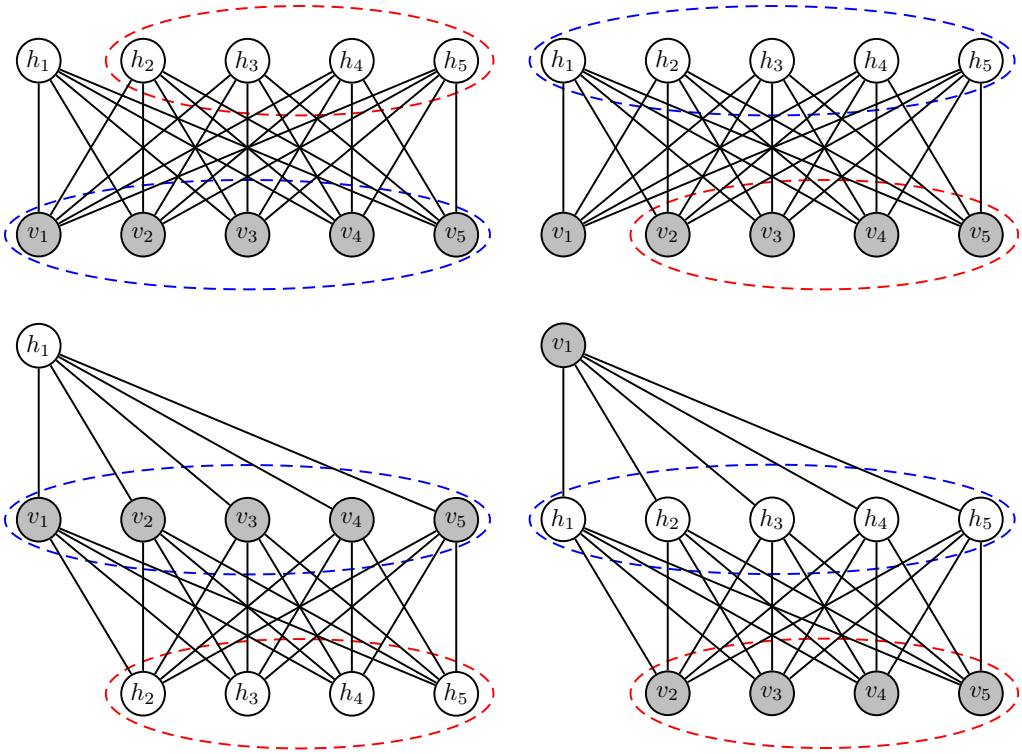


Figure 2.1: Conditional independence on an MRF: Conditional on the nodes within the blue ellipse, the other nodes become independent of each other. Two example separations (columns) are shown with two visualizations each (rows).

Boltzmann distribution as follows:

$$p(\mathbf{x} \mid \theta) = \frac{1}{Z} \exp\{-E(\mathbf{x}; \theta)\}, \tag{2.3}$$

where E denotes the energy-function on the random variables in the model. The parameters of the energy-function θ define the specific form of the energy-function, and parametrize the model. As before, Z is the partition function.

In the following we will briefly review undirected graphical models closely related to the work in the thesis. See Koller and Friedman [2009] for more details on generic properties of the models, other specific models, and further details. We start by discussing MRFs with observed random variables only, followed by a discussion on MRFs with hidden units, with a focus on energy-based models. We will then consider parameter sharing methods for learning large image models, and techniques for building deep belief networks. Finally, we describe MRFs proposed for generative image modelling.

2.1.1 Fully visible models

The Gaussian Markov random field (GMRF)-model is a classical, yet still often useful fully visible MRF. The joint probability distribution of the observations under a GMRF follows a multivariate Gaussian, with some mean vector and a precision matrix. The model can be graphically illustrated by drawing links between visible nodes for which the corresponding precision matrix element is non-zero. In other words, the lack of a connection between two units in a graphical model of a GMRF means the corresponding element in the precision matrix is zero, and vice versa. The model can be easy to interpret, and has several computationally attractive properties. However, it is not a very flexible model as it defines only a uni-modal distribution over the set of observations (simply because the Gaussian distribution is unimodal, independent on the number of dimensions it has). If the target density is multi-modal, a Gaussian can end up wasting significant amounts of probability mass in regions between data density modes and for example also outside of them to cover the modes sufficiently well, where the data distribution might not have any probability mass.

In product-of-experts (PoE) models [Hinton, 2002b], the joint probability of data $\mathbf{v}$ given model parameters θ can be written as a normalized product of potentially un-normalized strictly positive potential functions $f_\alpha(\mathbf{v}; \theta_\alpha)$, or experts,

as follows:

$$p(\mathbf{v} \mid \theta) = \frac{1}{Z} \prod_{\alpha} f_\alpha(\mathbf{v}; \theta_\alpha). \tag{2.4}$$

Note that setting the potential functions to be strictly positive functions acting on the cliques of the graph, one obtains the generic formulation of MRFs (see equation (2.1)). The models can be expressed using the energy-based notation by defining

$$E(\mathbf{v}; \theta) = -\log \left\{ \prod_{\alpha} f_\alpha(\mathbf{v}; \theta) \right\} = -\sum_{\alpha} \log f_\alpha(\mathbf{v}; \theta_\alpha), \tag{2.5}$$

and requiring each $f_\alpha(\cdot; \theta_\alpha)$ to be strictly positive. Therefore the PoE is widely applicable, and encapsulates for example the GMRF model.

2.1.2 Partially observed models

Partially observed MRFs include hidden units. It is often natural to assume the latent variables interact with the observed variables in a certain way. In undirected graphical models the causality in the interaction needs not to be specified, which might help in the model description. Including latent variables in the model might yield computational and representational benefits, as mentioned earlier. In particular, they might induce high-order dependencies with a small number of parameters, and reveal latent causes or features interpretable to the modeler.

In the following we will be reviewing MRFs with hidden variables, focusing on a class of models called restricted Boltzmann machines (RBMs) [Hinton, 2002a, Freund and Haussler, 1992, Smolensky, 1986]. Their conditional dependency structures under the joint model of hidden units and visible units are relatively simple: a unit within one of the two sets of unit types conditional on the units on the other set type is independent on the other units on the own type, as illustrated in Figure 2.1.

In general Boltzmann machines [Hinton and Sejnowski, 1983, Ackley, Hinton, and Sejnowski, 1985], connections can be/are allowed within the two sets of random variables, and the 'restricted' in the name of RBMs comes from the fact that the connectivity of the Boltmann machine is restricted to not have such connections. Other restricted forms of a general Boltzmann machine include the so-called deep Boltzmann machines [Salakhutdinov and Hinton, 2009] in which

there is less restriction on the connectivity between the hidden units, and the so-called semi-restricted Boltzmann machines [Osindero and Hinton, 2008], which define connectivities between the visible units explicitly. In several RBMs described below and used in the thesis, there are connectivities between the visible units, but the structure is not easily described via direct/graphical means, as it is dependent on the hidden unit configuration.

2.1.2.1 Restricted Boltzmann machines

Restricted Boltzmann machines have a bi-partite structure between the hidden and visible unit sets. They can be used as effective means in building deep belief networks [Hinton et al., 2006, Bengio, 2009], and also have shown significant promise in statistical modelling of image data [Bengio et al., 2013]. In the following, we will briefly describe them, and their translation equivariant extensions. See Bengio [2009] for a recent in-depth review on these models.

2.1.2.2 Bi-partite models based on harmoniums

RBMs, also known in the literature as harmoniums and combination machines, are generative probabilistic models consisting of a *bi-partite* set of random variables: visible units (observations) $\mathbf{v}$, and hidden units $\mathbf{h}$, and in the case of binary units, they are parametrized by their respective biases $\mathbf{a}$ and $\mathbf{b}$, and connections weights $\mathbf{W}$ between the partitions. As in fully-visible Boltzmann machines, the joint probability of the random variables conditional on model parameters $\theta = \{\mathbf{a}, \mathbf{b}, \mathbf{W}\}$ is given by the Boltzmann-distribution

$$p(\mathbf{v}, \mathbf{h}|\theta) \propto \exp\{-E(\mathbf{v}, \mathbf{h}|\theta)\}, \tag{2.6}$$

where the energy function $E(\mathbf{v}, \mathbf{h}|\theta)$ for any RBM model is *bi-linear* due to the bi-partite structure. Assuming binary units, this energy function can be written as follows:

$$E(\mathbf{v}, \mathbf{h}|\theta) = -\sum_i v_i a_i - \sum_j h_j b_j - \sum_i \sum_j W_{ij} v_i h_j. \tag{2.7}$$

An extension to continuous visible units can be obtained with the following energy-function:

$$E(\mathbf{v}, \mathbf{h}|\theta) = \sum_i \frac{1}{2\sigma_i^2}\left(v_i^2 - 2a_i v_i\right) - \sum_j h_j b_j - \sum_i \frac{1}{\gamma_i} \sum_j W_{ij} v_i h_j. \tag{2.8}$$

where σ_i is a scalar model parameter, and $\gamma_i = \sigma_i$ [Hinton and Salakhutdinov, 2006]. In Cho et al. [2011] $\gamma_i = \sigma_i^2$, which has advantageous properties compared to the former in learning, as will be shown shortly.

Due to lack of intra-layer dependencies, the joint conditional distributions of these models factorize over sites:

$$p(\mathbf{v}|\mathbf{h},\theta) = \prod_i p(v_i|\mathbf{h},\theta) = \prod_i \exp\left\{\langle \alpha, \phi(v_i,\mathbf{h})\rangle - A_i(\alpha)\right\} \tag{2.9}$$

$$p(\mathbf{h}|\mathbf{v},\theta) = \prod_j p(h_j|\mathbf{v},\theta) = \prod_j \exp\left\{\langle \beta, \phi(h_j,\mathbf{v})\rangle - B_j(\beta)\right\}, \tag{2.10}$$

where in the latter expressions, the conditional distributions are written in an exponential family form, where α, and β denote the canonical parameters of the distributions, ϕ denotes the vector of sufficient statistics, and A and B denote the log-normalizers of their partition functions. In this case of binary units, the conditional distributions are Bernoulli-distributed. In the case of the energy function of (2.8), the conditional probability distributions of visible units are univariate Gaussians, such that

$$p(v_i \mid \mathbf{h},\theta) = \mathcal{N}\left(v_i; a_i + \frac{\sigma_i^2}{\gamma_i}\sum_j W_{ij}h_j, \sigma_i^2\right). \tag{2.11}$$

Notice that for the expression of Cho et al. [2011], in which $\gamma_i = \sigma_i^2$, the means of the Gaussians are independent on the σ_i-parameters, which is not the case when $\gamma_i = \sigma_i$ as in Hinton and Salakhutdinov [2006].

The general class of harmoniums consistent with the above formulation (of (2.9) and (2.10)) are referred to in the literature as exponential family harmoniums [Welling et al., 2004]. Harmoniums are also examples of the PoE-models described earlier. For the binary unit harmonium of above, the hidden units can be integrated out, giving

$$\begin{aligned} p(\mathbf{v} \mid \theta) &= \sum_{\mathbf{h}} \frac{1}{Z}\exp\left\{-E(\mathbf{v},\mathbf{h}|\theta)\right\} \\ &= \frac{1}{Z}\exp\left\{\sum_i v_i a_i\right\}\sum_{\mathbf{h}}\exp\left\{\sum_j h_j(b_j + \sum_i W_{ij}v_i)\right\} \qquad (2.12) \\ &= \frac{1}{Z}\exp\left\{\sum_i v_i a_i\right\}\prod_j\sum_{h_j}\exp\left\{h_j(b_j + \sum_i W_{ij}v_i)\right\} \\ &= \frac{1}{Z}\underbrace{\exp\left\{\sum_i v_i a_i\right\}}_{f_0(\mathbf{v};\theta)}\prod_{j=1}^{J}\underbrace{\left(1+\exp\left\{b_j + \sum_i W_{ij}v_i\right\}\right)}_{f_j(\mathbf{v};\theta)}. \qquad (2.13) \end{aligned}$$

It is often convenient to describe an energy-based model via its free-energy F, which describes the energy of the Boltzmann distribution that is associated with the joint marginal distribution of the visible units (which for models with hidden units is obtained by integrating them out), as suggested already. Therefore the general expression to obtain the free-energy for discrete hidden units is

$$\mathrm{F}(\mathbf{v} \mid \theta) = -\log \left\{ \sum_{\mathbf{h}} \exp \left\{ -\mathrm{E}(\mathbf{v}, \mathbf{h}) \right\} \right\},$$

and for continuous hidden units,

$$\mathrm{F}(\mathbf{v} \mid \theta) = -\log \left\{ \int \exp \left\{ -\mathrm{E}(\mathbf{v}, \mathbf{h}) \right\} \, \mathbf{dh} \right\}.$$

Although maximum likelihood learning for most practical harmoniums is intractable[1], effective learning algorithms for learning them, and other restricted Boltzmann machines have been proposed, as we will describe later on. One very beneficial property of these bi-partite models is that efficient blocked Gibbs sampling schemes are available, for both learning and inference procedures, due to their conditional independence properties.

2.1.2.3 Boltzmann machines as mixture models

Boltzmann machines with (more than one) hidden units define a (highly structured) mixture model:

$$p(\mathbf{v}) = \begin{cases} \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}) = \sum_{\mathbf{h}} p(\mathbf{h}) \, p(\mathbf{v}|\mathbf{h}) \,, & \text{[discrete hiddens]} \\ \int p(\mathbf{v}, \mathbf{h}) \; d\mathbf{h} = \int p(\mathbf{h}) \, p(\mathbf{v}|\mathbf{h}) \; d\mathbf{h}, & \text{[continuous hiddens]}. \end{cases}$$

In most of the Boltzmann machine models for natural image data the mixture components $p(\mathbf{v}|\mathbf{h})$ are Gaussian, i.e. $p(\mathbf{v} \mid \mathbf{h}) = \mathcal{N}(\mathbf{v}; \mu, \Lambda^{-1})$. Below, the typical structures of these Gaussians are such that the hidden units control i) the means of Gaussians with diagonal precision matrices (as for example in the Gaussian-Bernoulli RBM of section 2.1.2.4) ii) the precision matrix structure of zero-mean Gaussians (as for example in the Product of Student-t experts model [Osindero et al., 2006] of section 2.1.2.5, and in the cRBM-model [Ranzato et al., 2010a] of section 2.1.2.6) or iii) both the mean, and the precision matrix structure (as for

[1] This is related to the fact that the partition function which normalizes the harmonium distribution enumerates over the state space of units, which grows exponentially with respect to their number.

example in the mPoT [Ranzato et al., 2010b] of section 2.1.2.8, and mcRBM [Ranzato and Hinton, 2010] of section 5.2.1).

The joint conditional distributions of all of these models are thus Gaussian Markov random fields (GMRF), with connections/edges between visible units i and k, if and only if $\Lambda(i,k) \neq 0$ (by definition). In the case of the Gaussian-Bernoulli RBM, there are no edges between the sites ($\Lambda(i,k) = 0 \ \forall \ i \neq k$), and the units are conditionally independent. For the other models, the existence of edges depends on the learned parameters and the hidden unit configuration, which affect the precision matrix (structure). In the following three subsections we will describe all of these models in more detail.

2.1.2.4 Gaussian-Bernoulli RBM

As described earlier, the binary hidden units $\mathbf{h}^m$ of the model are independent and distributed according to a Bernoulli distribution, conditional on the visible units $\mathbf{v}$. Conditional on hidden units $\mathbf{h}^m$, the visible units have an independent Gaussian distribution according to (2.11). It is important to note that the precision matrix of the conditional distribution is diagonal (because the units are conditionally independent), and the model places significant focus on modelling the means of these distributions. Having the possibility to have different means is useful for data containing multiple modes. However, using spherical components, and especially with fixed variances as commonly used, many components are needed for effective modelling performance. However, the exponential nature of the RBM model latent state space can help in this regard.

2.1.2.5 PoT

The Product of Student-t Experts (PoT) [Osindero et al., 2006] model belongs to the Product-of-Experts (PoE) framework, with generalized Student-t form experts:

$$f_j(y_j;\theta) \propto \frac{1}{(1+\frac{1}{2}y_j^2)^{\gamma_j}},$$

where the model parameters $\theta = \{\gamma_j > 0, \mathbf{K}\}$, and $\mathbf{y}$ is a linear transform of observed data $\mathbf{v}$, such that $y_j = \sum K_{ij} v_i$. The free-energy under the model is thus

$$\mathrm{F}(\mathbf{v} \mid \theta) = \sum_j \gamma_j \log\left\{1 + \frac{1}{2}\left(K_{\cdot j}{}^{\top}\mathbf{v}\right)^2\right\}. \tag{2.14}$$

The PoT can be formulated with auxilliary continuous-valued hidden units $\mathbf{h}^c$ [Osindero et al., 2006], leading to the energy function:

$$\mathrm{E}_{\mathrm{PoT}}(\mathbf{v}, \mathbf{h}^c) = \sum_j h_j^c \left(1 + \frac{1}{2}\left[\mathbf{K}_{\cdot j}{}^\top \mathbf{v}\right]^2\right) + (1 - \gamma_j) \log h_j^c. \tag{2.15}$$

See Appendix A.3 for a derivation of the free-energy from the above energy function. Conditional on the visible units, the hidden units are independent with Gamma distributions. Conditional on the hidden units, the visible units are distributed jointly according to a zero-mean multivariate Gaussian: $p(\mathbf{v} \mid \mathbf{h}^c) = \mathcal{N}\left(\mathbf{v}; \mathbf{0}, \Lambda = \left(\mathbf{K}\,\mathrm{diag}\left\{\mathbf{h}^c\right\}\mathbf{K}^\top\right)^{-1}\right)$, where Λ denotes the covariance matrix, $\mathrm{diag}\left\{\mathbf{h}^c\right\}$ denotes a diagonal matrix with the elements of $\mathbf{h}^c$ in the diagonal, and $\mathbf{K}$ is a matrix of size the number of visible units times the number of hidden units (or the experts). In contrast to the Gaussian-Bernoulli RBM, the visible units can be coupled in their joint distribution conditional on the hidden units because the precision matrix is not restricted to be diagonal, with the structure being dependent on the hidden unit activation pattern. Highly correlated data-dimensions are thus not as problematic as with the GB-RBM. As the mean and mode of the model is fixed to zero and the components are symmetric about it, data densities that are not symmetric about the origin, and those with modes far away from the origin, are some examples of potentially problematic distributions for the PoT to model.

2.1.2.6 cRBM

Ranzato, Krizhevsky and Hinton [Ranzato et al., 2010a] develop an extension of RBMs in which binary hidden units modulate pairwise interactions between visible units, according to the following energy function:

$$\mathrm{E}(\mathbf{v}, \mathbf{h} \mid \theta) = -\sum_i \sum_k \sum_j v_i v_k h_j^c W_{ik}^j - \sum_j d_j h_j^c = -\sum_j \left(d_j + \mathbf{v}^\top W^j \mathbf{v}\right) h_j^c, \tag{2.16}$$

where the model parameters $\theta = \{\mathbf{d}, \mathbf{W}\}$, where the weights $\mathbf{W}$ are factorized such that each weight is a sum of product of three terms $W_{ik}^j = \sum_f K_{if} K_{kf} \pi_{jf}$. The free-energy of this cRBM then becomes

$$\mathrm{F}(\mathbf{v} \mid \theta) = -\sum_j \log\left\{1 + \exp\left\{d_j + \sum_f \pi_{jf}\left(\mathbf{v}^\top K_{\cdot f}\right)^2\right\}\right\}, \tag{2.17}$$

meaning that the model is a PoE with experts

$$f_j(\mathbf{v};\theta) = 1 + \exp\left\{ d_j + \sum_f \pi_{jf} \left(\mathbf{v}^\top K_{\cdot f}\right)^2 \right\}.$$

As in the PoT-model, the joint probability distribution of visible units conditional on the hidden units is a zero-mean multivariate Gaussian, here with precision matrix $\Lambda^{-1} = \mathbf{K}\text{diag}(\pi\mathbf{h}^c)\mathbf{K}^\top$.

2.1.2.7 mcRBM

The mcRBM [Ranzato and Hinton, 2010] extends the cRBM to more general means. The model is constructed by combining the energy-functions of the GB-RBM and the cRBM, with shared visible units, but different hidden units for the two parts:

$$\mathrm{E}_{\mathrm{mcRBM}}(\mathbf{v}, \mathbf{h}^c, \mathbf{h}^m) = \mathrm{E}_{\mathrm{GB-RBM}}(\mathbf{v}, \mathbf{h}^m) + \mathrm{E}_{\mathrm{cRBM}}(\mathbf{v}, \mathbf{h}^c). \tag{2.18}$$

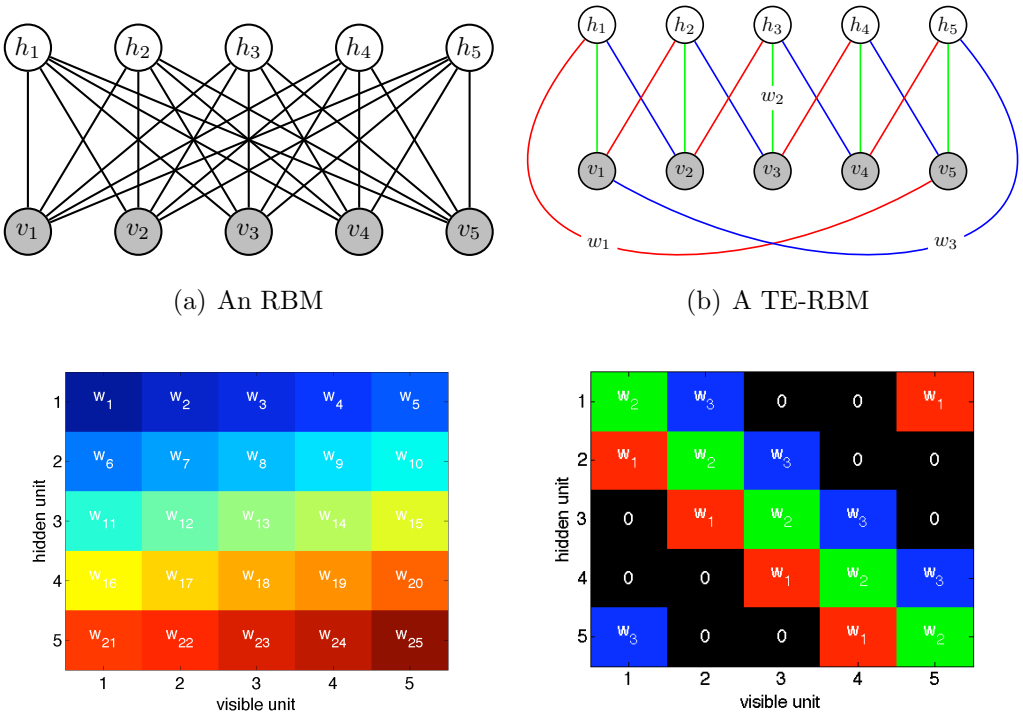
Conditional on the visible units, the hidden units $\mathbf{h}^c$, and $\mathbf{h}^m$ are independent and distributed according to Bernoulli distributions, as in the two models separately. Conditional on the hidden units, the visible units are distributed jointly as multivariate Gaussians with an adjustable mean vector and precision matrix, both being dependent on the hidden unit configurations.

2.1.2.8 mPoT

The mPoT [Ranzato et al., 2010b] extends the PoT in a similar way the mcRBM extended the cRBM: the model is constructed by combining the energy-functions of the GB-RBM and the PoT, with shared visible units, but different hidden units for the two parts:

$$\mathrm{E}_{\mathrm{mPoT}}(\mathbf{v}, \mathbf{h}^c, \mathbf{h}^m) = \mathrm{E}_{\mathrm{GB-RBM}}(\mathbf{v}, \mathbf{h}^m) + \mathrm{E}_{\mathrm{PoT}}(\mathbf{v}, \mathbf{h}^c). \tag{2.19}$$

Conditional on the visible units, the hidden units $\mathbf{h}^c$, and $\mathbf{h}^m$ are independent and distributed according to Gamma, and Bernoulli distributions, as in the two models separately. Conditional on the hidden units, the visible units are distributed jointly as multivariate Gaussians. Assuming $\sigma_i = \gamma_i = \sigma$ in the GB-RBM energy as defined in 2.8, we have that $p(\mathbf{v} \mid \mathbf{h}^c, \mathbf{h}^m) = \mathcal{N}\left(\mathbf{v}; \Lambda\left(\mathbf{a} + \sigma\mathbf{W}\mathbf{h}^m\right), \Lambda\right)$, where $\Lambda = \left(\mathbf{K}\text{diag}\left\{\mathbf{h}^c\right\}\mathbf{K}^\top + \sigma^{-2}\mathbf{I}\right)^{-1}$, and $\mathbf{W}$ denotes the weights of the GB-RBM. Also in this model the mean can be non-zero, and the precision can be non-diagonal, being dependent on the hidden unit assignments.



(a) An RBM (b) A TE-RBM

(c) Weight matrix for the RBM in (a) (d) Weight matrix for the TE-RBM in (b)

Figure 2.2: Graphical comparison of a restricted Boltzmann machine (RBM), and a translation-equivariant RBM (TE-RBM). In an RBM, each hidden unit h_j is connected to all of the visible units v_i, with unit-specific connectivity weights. In the TE-RBM, hidden units are connected typically only to a finite subset of visible units, specified by a receptive field system, with a universally shared set of weights. In this TE-RBM example, the subsets all contain three visible units, and the respective connectivity weights are encoded with different colours.

2.1.3 Translation equivariant extensions: Convolutional models

Translation equivariant RBMs (here abbreviated as TE-RBMs) assume a more restrictive parametrization than a regular RBM, typically involving a reduced connectivity structure. To achieve translation equivariance, the biases need to be layer-specific so that each unit in a layer shares the same bias, and the hidden units within a layer need to connect to the visible units with layer-specific connection weights. Typically each hidden unit in a TE-RBM is connected only to a subset (of fixed cardinality) of the visible units, specified by a receptive field system. In the example TE-RBMs of Figure 2.2 (b), each hidden unit is

connected to three of the five visible units with a set of shared kernel of connectivity weights $\omega = \{\omega_1, \omega_2, \omega_3\}$. To address the issue of finite length unit layers while maintaining translation-equivariance, the model assumes a (toroidal) wrap-around. The full connectivity weight matrix $\mathbf{W}$ between a hidden layer and the visible units layer, with elements $W(j,i)$ representing the weight between hidden unit j and visible unit i, is circulant[2], with zero-weights assigned to disconnected unit pairs. Weight-matrices for the RBM and TE-RBM models of Figure 2.2(a) and Figure 2.2(b) are shown in Figure 2.2(c) and Figure 2.2(d), respectively.

Thus a TE-RBM is completely specified by a receptive field system, a kernel of connection weights ω for each hidden unit – visible unit layer pair, and layer-specific unit biases. In a binary-unit TE-RBM with multiple feature layers (indexed by α), the model energy can be decomposed as follows:

$$\begin{aligned} E(\mathbf{v}, \mathbf{h}|\theta) &= -a \sum_i v_i - \sum_\alpha b_\alpha \sum_j h_{\alpha j} - \sum_\alpha \sum_j h_{\alpha j} \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha_{d(j,\ell)} \\ &= -a \sum_i v_i - \sum_j \sum_\alpha h_{\alpha j} \left(b_\alpha + \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha_{d(j,\ell)} \right), \end{aligned} \tag{2.20}$$

where a denotes the visible layer bias, b_α denotes the bias for hidden layer α, $\mathrm{N}_{\alpha j}$ contains the indices of the visible units within the receptive field of hidden unit j at feature layer α, and $\omega^\alpha_{d(j,\ell)}$ is the weight between that hidden unit and visible unit ℓ, where $d(a,b)$ denotes a map from unit indices to kernel weight indices.

See Figure 2.3 for a graphical illustration of a generic TE-RBM, and Figure 2.4 for a simple TE-RBM with two feature planes for two-dimensional data. These translation-equivariant models discussed above are often called convolutional in the literature (as for example in Lee et al. [2009]).

Clearly the marginal distributions of the models can be written in a product-of-experts form, with site-independent experts f_α:

$$p(\mathbf{v} \mid \theta) = \frac{1}{Z} \exp \left\{ a \sum_i v_i \right\} \prod_{j=1}^{J} \prod_\alpha f_\alpha(\mathbf{v}_{\mathrm{N}_{\alpha j}}; \theta), \tag{2.21}$$

where sites are indexed by j. In this homogeneous PoE-representation of the TE-RBM, the experts are given by

$$f_\alpha(\mathbf{v}_{\mathrm{N}_{\alpha j}}; \theta) = 1 + \exp \left\{ b_\alpha + \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha_{d(j,\ell)} \right\}. \tag{2.22}$$

[2] $W(j,i) = c(1 + (i-j) \bmod I)$, where $\mathbf{c}$ is a vector of length I representing weights from hidden unit h_1 to all (I) of the visible units $v_1, \ldots, v_I$ in the visible unit layer, such that $c(i)$ represents the weight between h_1 and v_i.

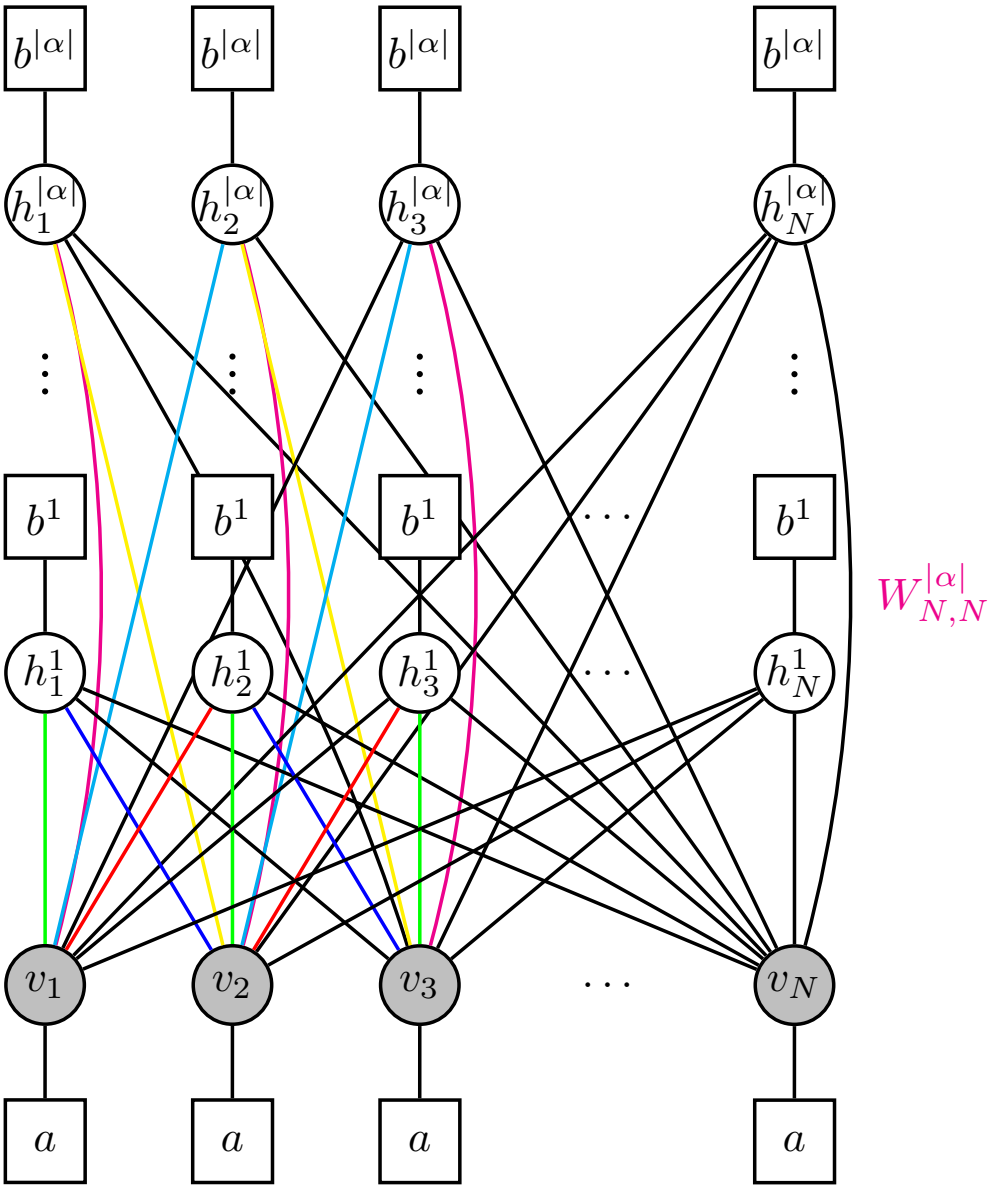


Figure 2.3: A TE-RBM with many feature planes (2 shown), each by definition having their own weight-kernels and biases, which are shared across the hidden unit sites within the planes. In the above graph, three different weight kernel elements have been colour-coded for both of the layers, shown for the connections between the first three hidden units in both layers and the first three visible units. Zero-valued weights (effectively meaning no connectivity) between visible and hidden units have also been drawn.

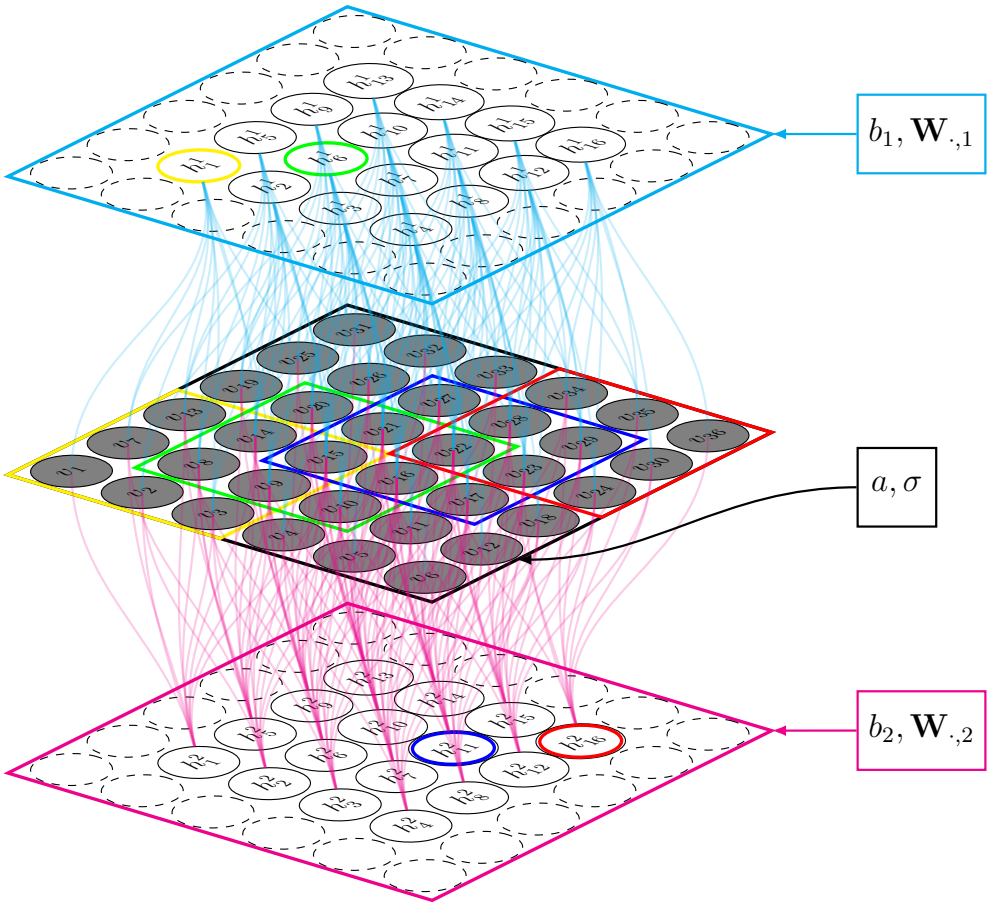


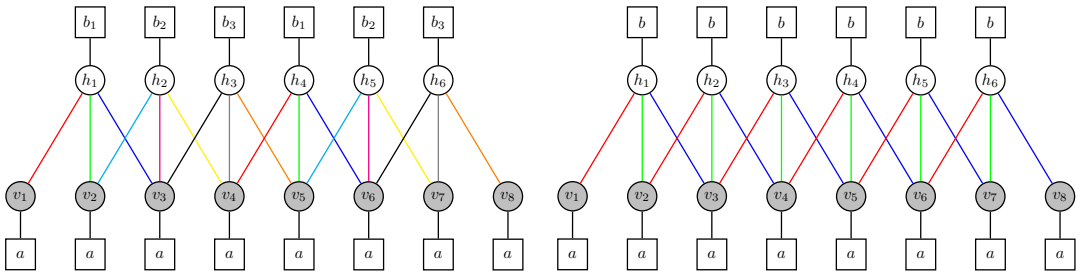
Figure 2.4: An illustration of the connectivity structure of a convolutional Gaussian RBM having two feature planes each of which have sixteen hidden units, each connected to nine visible units. For both of the feature layers two hidden units as well as their receptive fields are highlighted. Each of the feature planes is associated with a filter and a bias, used as the parameters for each of the hidden units within the feature plane. Although the connections have been drawn with the same colours, their weights specified by the filters will typically have different values. The visible unit layer is associated with bias a and σ parameters.

Without parameter sharing of any kind, RBMs for large visible unit sets, such as for large images, have too many free parameters to be learned, and such models have been mainly used as models for image patches. Tying the parameters in a translation equivariant way enables considering models with large(r) number of units, and this has been one way to scale beyond patches. The use of such parameter tying is often justified in the context of modelling natural images by stationarity in the image data statistics.

The spatial support of images is however not infinite, and images are not toroidal (as assumed by the default wrap-around boundary handling) at least in the typical cases, and the mismatch in the data and the model structures due to inexact boundary assumptions can cause issues. Another obvious downside of the parameter reduction is that the model becomes less flexible, and potentially unable to capture the statistics existing in the data to a sufficient degree.

The type of feature sharing has been used for developing several translation equivariant Boltzmann machines, including convolutional Gaussian RBMs [Lee et al., 2009], PoT-models [Roth and Black, 2005], and the Bi-FoE model [Heess et al., 2009]. It will be also used in the following chapter as providing translation equivariance to the further equivarianced models that will be developed.

Tiled-convolutional parameter sharing (see for example Ranzato et al. [2010b]) is an another approach for reducing parametrization and obtaining translation equivariance. Figure 2.5 compares two RBMs, one with convolutional, and the other with tiled-convolutional parameter sharing. On the left of the figure, a graphical representation of a tiled-convolutional RBM, and its weight matrix are depicted. Similar to the convolutional model on the right, each hidden unit is connected to a subset of visible units (here three). Whereas in the convolutional model hidden units at different spatial offsets have the same parameters, in the tiled-convolutional model parameters are shared in blocks. Here the block size is three hidden units, and the hidden unit sets from location 1 to 3, and the set from 4 to 6 are associated with the same parameters. Notice that in this example there are three times as many parameters per hidden unit layer in the tiled-convolutional model when compared to the convolutional model. In both of the cases the visible unit biases have been tied across the layer. The 2D equivalent of this parameter sharing scheme is obtained simply by considering rectangular receptive fields for each hidden unit. Diagonally-tiled-convolutional parameter sharing uses only the hidden units within the diagonals of the blocks,



(a) A tiled-convolutional RBM (b) A TE-RBM

Figure 2.5: Graphical comparison of a translation-equivariant RBM (TE-RBM), and tiled-convolutional RBM. The hidden units of the models are connected to a subset of visible units, specified by a receptive field system. In the TE-RBM a single parameter kernel of weights and a bias is shared across the hidden unit lattice. In the tiled-convolutional RBM, a parameter kernel is shared across the hidden unit lattice in a block-manner.

or a further thinned subset of them by using a fixed offset. Figures 2.6 and 2.7 illustrate graphically two diagonally-tiled-convolutional Gaussian-Bernoulli RBM instances.

2.1.4 Hierarchical extensions: Deep belief nets

Deep belief networks (see Bengio [2009] for a recent in-depth review) model observations using a cascade of inter-connected layers of random variables, from the visible unit layer $\mathbf{v}$ to the L^{th} hidden unit layer $\mathbf{h}^L$. Conditional on the parameters of the DBN consisting of the intra-connection weights, and the biases, the joint probability of the random variables in the model factorizes over layer-wise distributions:

$$p(\mathbf{v}, \mathbf{h} \mid \theta) = p\big(\mathbf{v} \mid \mathbf{h}^1, \theta^1\big) \left(\prod_{\ell=1}^{L-2} p\big(\mathbf{h}^\ell \mid \mathbf{h}^{\ell+1}, \theta^{\ell+1}\big) \right) p\big(\mathbf{h}^{L-1}, \mathbf{h}^L \mid \theta^L\big) \,. \qquad (2.23)$$

Deep belief nets have a chain graph representation, with undirected connections from the top two layers L and $L-1$, and top-down directed connections between the layers below. Efficient learning algorithms utilizing the factorization property have been developed to learn model parameters greedily by an incremental bottom-up training of layer-wise harmoniums, each of the layers $\ell = 2, \ldots, L$ using unit probabilities of the layer $\ell - 1$ as their training data. Fine-tuning can be then done by using back-fitting schemes [Hinton et al., 2006, Bengio, 2009].

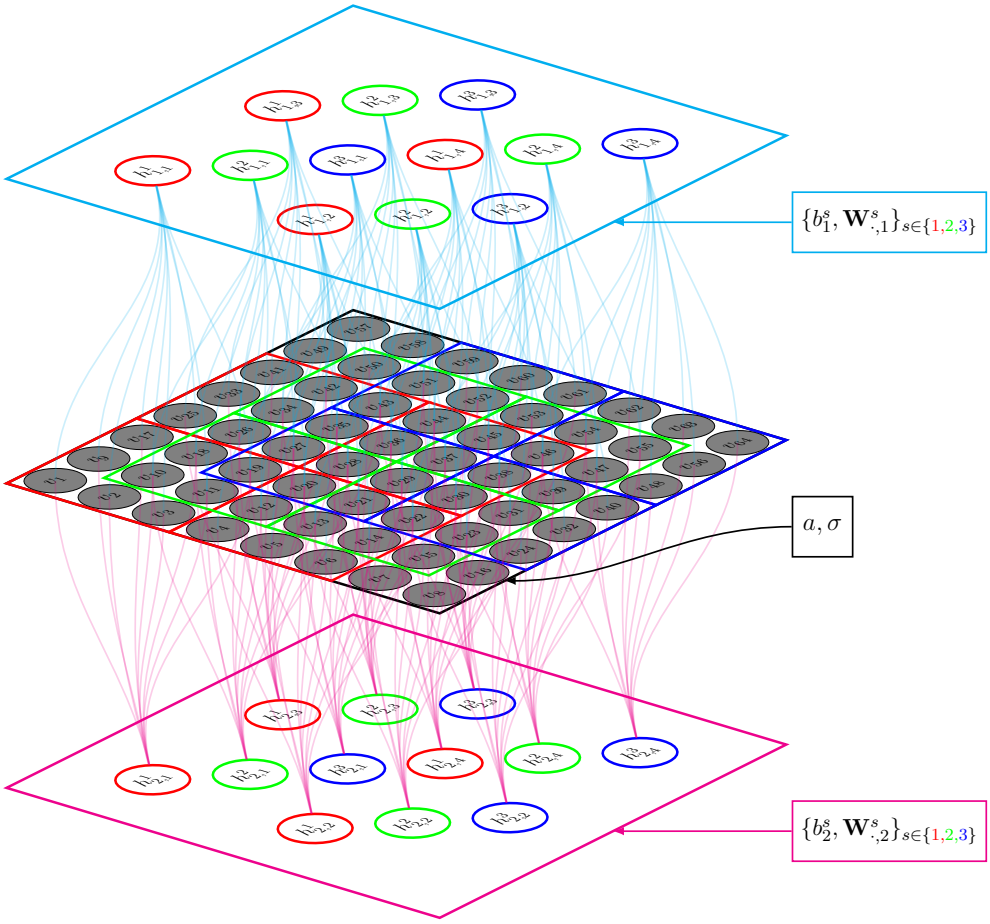


Figure 2.6: An illustration of the connectivity structure of a diagonally-tiled-convolutional Gaussian RBM with three sets of hidden units (red,green,blue), each having two feature planes each of which have four hidden units. Each of the feature planes are associated with a filter and a bias, used as the parameters for each of the hidden units within the feature plane. Under each of the sets, the hidden units connect to the visible units in a non-ovelapping manner, tiling a certain sized region. The connectivity structure is the same for each of the feature planes within a set. The hidden units under different sets connect to visible units in a similar manner, but are applied with an offset from each other by a diagonal shift of one unit between neighboring sets. The visible unit layer is associated with bias a and σ parameters.

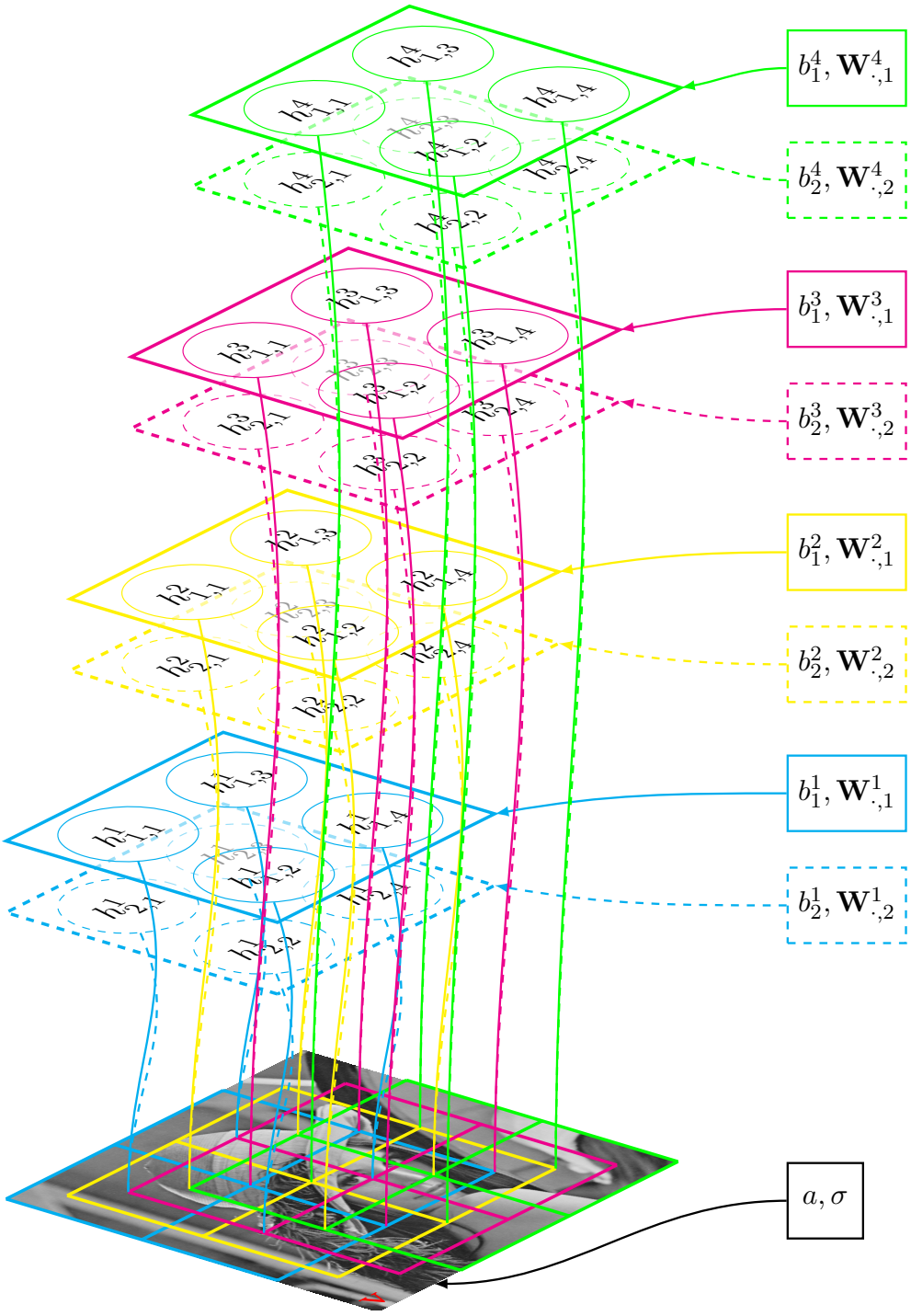


Figure 2.7: An illustration of the connectivity structure of a diagonally-tiled-convolutional Gaussian RBM with four sets of hidden units (associated with different colours), each having two feature planes each of which have four hidden units. Each of the feature planes are associated with a filter and a bias, used as the parameters for each of the hidden units within the feature plane. Under each of the sets, the hidden units connect to the visible units in a non-ovelapping manner, tiling a certain sized region. The connectivity structure is the same for each of the feature planes within a set. The hidden units under different sets connect to visible units in a similar manner, but are applied with an offset from each other by a diagonal shift. The visible unit layer is associated with bias a and σ parameters.

2.1.5 Generative models for image patches

There exists a wealth of generative models based on undirected and chain graphs (which contain both directed and undirected connections), many of them using product of experts (PoE)-models and their extensions as their building blocks. Many of these have conditionally Gaussian representations, as is also very popular amongst the models for wavelet decomposed images. Many of the conditionally Gaussian models mentioned earlier (for example the PoT, cRBM, and mcRBM models) were developed in the context of modelling image patches.

Deep belief nets [Hinton et al., 2006] are an example of models having a chain graph representation. Lee et al. [2008] develop a DBN with two layers for natural image patches by using contrastive divergence training with sparsity regularization to learn the layer-specific RBMs within the model. The bottom layer RBM is the Gaussian-Bernoulli model, whereas the top layer is a binary unit (Bernoulli) RBM. The PoT model is extended to topographical and hierarchical representations in Osindero et al. [2006]. Osindero and Hinton [2008] develop a deep architecture with lateral connections, based on semi-restricted Boltzmann machines, which extend RBMs to have connections within the visible unit layer. However, as mentioned by the authors, more powerful DBNs (models) can be obtained when the hidden units can modulate the pairwise interactions between the visible units (as for example in the PoT, and cRBM-based models).

2.1.6 Generative models for images

Most of the energy-based models that have been proposed in the literature for whole images are MRFs, with parameter sharing across sites, in either convolutional or tiled-convolutional manner. In the former case, they form homogeneous PoE-models, with a set of experts shared across sites. Denoting neighborhood of node i as $\mathrm{N}(i)$ (defined with a neighborhood system), we write the free-energy for such models as follows:

$$\mathrm{F}(\mathbf{v} \mid \theta) = -\sum_i \sum_j \log \left\{ f_j(\mathbf{v}_{\mathrm{N}(i)}; \theta) \right\}, \tag{2.24}$$

where f_j denotes the site-independent expert function (of index j). Koster et al. [2009] use experts defining logistic distributions on the filter responses of the neighborhood data[3]. In the Fields-of-Experts (FoE) model [Roth and Black, 2007]

[3]Via personal communication

the potential functions of the experts are un-normalized Student-t distributions

$$f_j(\mathbf{v}_{\mathrm{N}(i)};\theta) = \left(1 + \frac{1}{2}\left[\mathbf{K}_{\cdot j}{}^{\top}\mathbf{v}_{\mathrm{N}(i)}\right]^2\right)^{-\gamma_j},$$

where $\mathbf{K}_{\cdot j}$ is the j^{th} filter, and $\gamma_j > 0$, resulting in the following (free-)energy

$$\mathrm{F}(\mathbf{v} \mid \theta) = \sum_i \sum_j \gamma_j \log\left\{1 + \frac{1}{2}\left[\mathbf{K}_{\cdot j}{}^{\top}\mathbf{v}_{\mathrm{N}(i)}\right]^2\right\}. \tag{2.25}$$

Note that this basically defines a homogeneous PoE extension of the PoT-model.

Several other expert distributions have been proposed. For example, the Bi-modal FoE model [Heess et al., 2009] extends FoE experts to be bi-modal, with

$$f_j(\mathbf{v}_{\mathrm{N}(i)};\theta) = \left(1 + \frac{1}{2}\left[\left(\mathbf{K}_{\cdot j}{}^{\top}\mathbf{v}_{\mathrm{N}(i)} + b_j\right)^2 + d_j\right]^2\right)^{-\gamma_j}, \tag{2.26}$$

where b_j, d_j, and γ_j are scalar parameters. Weiss and Freeman [2007] on the other hand use discrete zero-mean Gaussian scale mixtures as the experts,

$$f_j(\mathbf{v}_{\mathrm{N}(i)};\theta) = \sum_{k=1}^{K} w_k \mathcal{N}\left(\mathbf{K}_{\cdot j}{}^{\top}\mathbf{v}_{\mathrm{N}(i)}; 0, \sigma_k^2\right), \tag{2.27}$$

where w_k denotes a mixing weight of k^{th} Gaussian mixture component, with variance σ_k^2. Ranzato et al. [2010b] develop tiled-convolutional extensions to the mcRBM, and the mPoT models.

Recently deep belief networks have been developed for whole images, using both translation equivariant and tiled-translation equivariant feature sharing. Both Lee et al. [2009] and Norouzi et al. [2009] propose translation equivariant DBNs by using parameter sharing in the layer-wise RBMs (TE-RBMs) which are used to construct the deep architectures. As has been found useful in the literature to learn localized filters, the models in the papers are learned by encouraging/forcing the feature layer activations to be sparse. Ranzato et al. [2011b] extend the mPoT to deep delief net architectures, using tiled-convolutional feature sharing for modelling image data.

It is clear that similar translation equivariant extensions could be developed for many other translation variant models discussed earlier. In any case, these models would deal image transformations equivariantly only with respect to translation, and not also with respect to other transformations, such as in-plane rotation. Memisevic and Hinton [2009] describe how to *learn* transformations based

on pairs of training images using factored 3-way Boltzmann machines. Such a network could be used in to identify for example rotated versions of a given pattern, e.g. by fixing a reference version of the pattern and inferring the transformation. However, it seems rather excessive to learn the machinery for this when it can be built in, as will be shown in the following chapter, where the main ideas related to this work will be outlined.

2.2 (Acyclic) Directed Graphical Models

In acyclic directed graphical models (DAGs) the connections between the nodes are directed, and are not allowed to form any directed cycles or loops. The directions can be interpreted as defining causality in the model. Acyclic directed graphical models with deterministic nodes are called feed-forward neural networks. With stochastic nodes the models are called Bayesian belief networks, or belief nets for short. Hybrid networks with both kinds of nodes have been also considered in the literature. Models with stochastic nodes are in principle more flexible than the ones with deterministic nodes, allowing e.g. one-to-many input-output mappings, and can perform explaining away.

For belief nets, the joint probability density of the J random variables $\{x_1, \ldots, x_J\}$ factorizes as a product of node-specific conditional probability distributions:

$$p(x_1, \ldots, x_J) = \prod_{j=1}^{J} p\big(x_j \mid x_{\mathrm{Pa}(x_j)}\big)\,, \tag{2.28}$$

where $\mathrm{Pa}(x_j)$ denotes the parents of node x_j. Marginally each node is dependent on its parent nodes, child nodes, and the parents on the child nodes, which define the so-called Markov blanket of the node [Koller and Friedman, 2009]. Recall in the case of the undirected models/MRFs, the local Markov property is defined based on only the directly connected nodes. However, belief nets can be converted onto MRFs, by the process of 'moralization'. For details on moralization, and for generic properties of the class of models, see Koller and Friedman [2009].

In the following we will only briefly review directed graphical models which are closely related to the work in the thesis, and applications of them for image modelling tasks. We start by describing image models developed in the computer vision community based on stochastic nodes/belief nets. In order to do this, we first discuss models for patches. The models for large images are

typically built on wavelet coefficients, and thus in describing the models we first describe wavelet decompositions in Sec. 2.2.1.2, and then models built on top of them in Sec. 2.2.1.3. Wavelet decompositions are widely used in the computer vision community in extracting features for early vision. We then discuss in Sec. 2.2.1.4 hierarchical image models based on compositional models developed to learn semantically higher level image representations. Finally in Sec. 2.2.2 we will describe feed-forward neural nets.

2.2.1 Generative Models for Image Data

2.2.1.1 Generative models of image patches

Motivated by a so-called efficient coding hypothesis, methods known as sparse coding or sparse components analysis aim to find an overcomplete linear basis optimal in terms of maximal sparseness and statistical independence of transform coefficients in response to natural images [Olshausen and Field, 1997]. These methods assume a linear observation model, where observed images $\mathbf{v}$ are noisy versions of a linear combination of basis functions/components $\mathbf{W}_{\cdot j}$, with associated mixing coefficients h_j:

$$v_i = \sum_j h_j W_{ij} + n_i, \tag{2.29}$$

where n is used to describe system noise. The probability of an image conditional on the basis functions is then

$$p(\mathbf{v} \mid \mathbf{W}) = \int p(\mathbf{h}, \mathbf{v} \mid \mathbf{W}) \; d\mathbf{h} = \int p(\mathbf{v} \mid \mathbf{h}, \mathbf{W})\, p(\mathbf{h}) \; d\mathbf{h}, \tag{2.30}$$

where $p(\mathbf{h}) = \prod_j p(h_j)$ is a factorial prior over the transform coefficients to incorporate statistical independence, and the distributions $p(h_j)$ are chosen from heavy-tailed, highly kurtotic distributions to encourage sparsity. These methods are closely related [Olshausen and Field, 1997] to the probabilistic versions of the independent component analysis (ICA) methods by Bell and Sejnowski [1996]. Obtained basis functions with these methods resemble the receptive fields of primary visual cortex, and also those of wavelet decompositions [Olshausen and Field, 1997]. Even though uncorrelated responses to natural images can be obtained with these bases, it is well known that strong residual dependencies typically remain [Wainwright et al., 2001, Simoncelli and Olshausen, 2001]. Motivated by the typically invalid assumption of independence, several approaches

have been proposed allowing dependence amongst the latent components, including the Independent Subspace Analysis (ISA) approach [Hyvarinen and Hoyer, 2000] and other hierarchical models such as those in Karklin and Lewicki [2003, 2005] and Garrigues and Olshausen [2008].

Tenenbaum and Freeman [2000] develop an extension of linear models with an additional set of variables, to factor image patches according to content (feature type), and style (transformation) with a *bilinear* function where

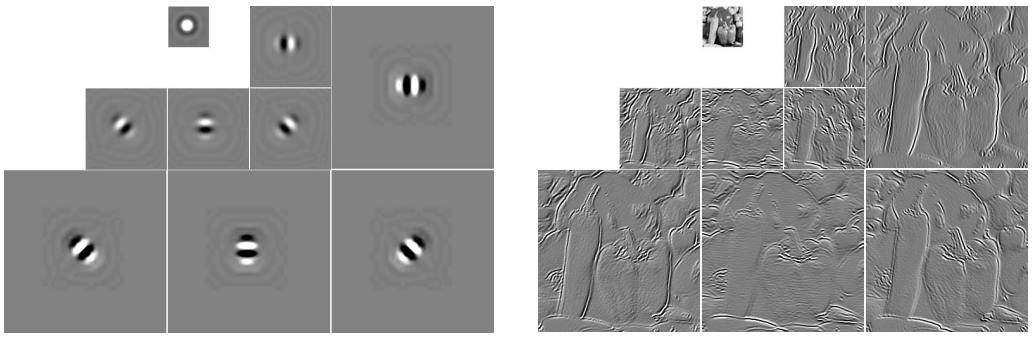
$$v_i = \sum_j \sum_k h_j z_k W_{ik}^j, \tag{2.31}$$

so that the basis function $\mathbf{W}_{\cdot k}^j$ that is associated with feature type j, and transformation k, is modulated by their respective coefficients h_j, and z_k. Grimes and Rao [2005] extend these ideas into a sparse coding framework to learn bilinear sparse components. See Olshausen et al. [2007] for similar models for image sequences. Several other generative models for image patches based on directed graphical models have been proposed, see Hyvarinen et al. [2009] for discussions on many of them. To scale onto large images, wavelet decompositions are often utilized which will be discussed in the following.

2.2.1.2 Wavelet decompositions

Wavelet decompositions define linear transformations of their input and can be written follows: $\mathbf{y} = \mathbf{A}\mathbf{x}$, where $\mathbf{x}$ denotes the input (a $I \times 1$-vector for a single input image/signal), $\mathbf{y}$ denotes the output (a $J \times 1$-vector), and $\mathbf{A}$ defines the particular decomposition (a $J \times I$-matrix, where $J \geq I$). The operation in practice performs filtering/convolving typically by a set of band-pass and low-pass filters, which are based on their respective canonical filter kernels by scaling/dilation and also rotation in the former case. Applying these filters, the classical orthogonal wavelet decompositions produce a set of (I) invertible transform coefficients $\mathbf{y}$, which can be partitioned onto so-called detail and scaling coefficients, and organized in a pyramid [Mallat, 1989]. The *fixed* set of band-pass basis functions (consisting of translated, dilated, and rotated versions of a common kernel) are qualitatively similar to the receptive fields of the primary visual cortex [Olshausen and Field, 1996]. To obtain transformation equivariance[4] into the representation, using *overcomplete* bases (where $J > I$) has been necessary [Simoncelli et al.,

[4]In a slightly weaker form termed shiftability.



(a) Basis functions illustration

(b) Natural image responses (high-pass residual not shown)

Figure 2.8: Illustration of basis functions for 3^{rd}-order steerable pyramids [Simoncelli et al., 1992], and the transform coefficients of a natural image (peppers). The top-left basis function corresponds to a low-pass filter/scaling function, whereas the other basis functions illustrated are oriented band-pass filters. The basis function images have been obtained by reconstructing a steerable pyramid with a single non-zero coefficient positioned at the center of the corresponding subband; also cropped and resized for visualization purposes. Figure credit: Kivinen et al. [2013a].

1992]. Although these transforms have benefits over the classical wavelet transforms for image analysis [Kivinen et al., 2007a,b, Portilla et al., 2003, Szeliski, 2010], the basis functions are fixed and the transform coefficients (see Figure 2.8 for an illustration of these for a transform instance) may not be always exactly invertible [Olshausen and Field, 1997] and then complicate the building of statistical models for *images*. In the following we will discuss models for the transform coefficients. See Szeliski [2010] for a recent review on wavelet transforms for more details on them.

2.2.1.3 Generative models for wavelet decomposed images

There are numerous efforts to capture the regularities existing in the decomposed images, ranging from local to global statistical analysis of the transform coefficients, and here we will describe only the most relevant to our work. See Srivastava et al. [2003] for a relatively recent review for a broader perspective, and also Simoncelli [2005], which focuses more on wavelet-based models. The marginal distributions associated with the 'detail' wavelet coefficients typically have high kurtosis and "heavy tails" when compared to a Gaussian (see Figure 2.9 for an

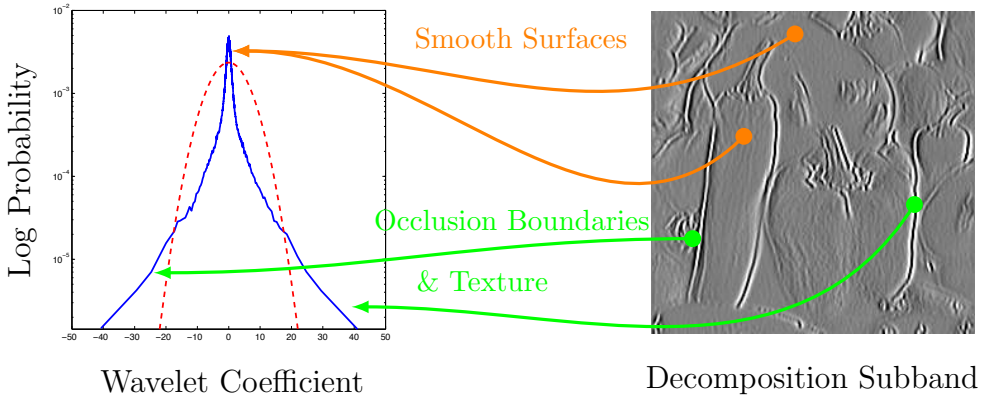


Figure 2.9: Empirical histogram (blue solid line) of a natural image subband coefficient values with maximum likelihood Gaussian fit overlaid (red dashed line). Figure credit: Kivinen et al. [2013a].

illustration for a typical case). These findings have motivated the development of wavelet marginal models based on generalized Gaussians[5] [Mallat, 1989, Srivastava et al., 2003], and mixture distributions. One widely used [Simoncelli, 2005] *continuous* mixture is the *Gaussian scale mixture* (GSM). These model a wavelet coefficient v_i as the product of two independent variables:

$$v_i = \sqrt{h_i} s_i \qquad h_i \geq 0,\ s_i \sim \mathcal{N}(0, \Lambda)\,. \tag{2.32}$$

Integrating out the scalar multiplier h_i produces the marginal density, which mixes Gaussians of varying scales:

$$p(v_i) = \int \mathcal{N}\left(v_i; 0, \Lambda\right) dG(\Lambda). \tag{2.33}$$

There are several mixing distributions $G(\Lambda)$ for these continuous mixtures good for modelling the marginal statistics [Wainwright et al., 2001, Kivinen et al., 2007b]. However, considering marginal statistics is typically not sufficient. In particular, wavelet decomposed images typically lead to coefficients containing dependencies, which is especially the case with overcomplete bases [Willsky, 2002, Srivastava et al., 2003]. Typical behaviour of large-magnitude detail coefficients persisting through scales and clustering at nearby locations can seen in Figure 2.8 (right). To model joint dependencies, Portilla et al. [2003] model wavelet coefficients in independent local neighborhoods. The orientation adaptive GSM [Hammond and Simoncelli, 2008] augments this model with scalar latent variable to

[5] In which the probability of a wavelet coefficient v_i given model parameters $\theta = \{\sigma, \rho\}$ is $p(v_i \mid \theta) \propto \exp\{-|v_i/\sigma|^\rho\}$.

describe dominant local orientation, allowing for example modelling of oriented texture at arbitrary rotations.

In Lyu and Simoncelli [2009] the basic local GSM is extended to model wavelet subbands with homogeneous fields of Gaussian scale mixtures, where coefficient fields (as opposed to scalars) are modeled as element-wise products of two independent random variable fields, both described with homogeneous Gaussian Markov random fields. A global tree-structured model for wavelet statistics based on GSMs has been developed in Wainwright et al. [2001]. It models wavelet coefficients in a vector-form combining subband coefficients at a particular scale and position over all of the orientations. The vectors (for each position at each scale) are marginally distributed according to GSMs but form a tree of observations evolving according to the following random processes: A premultiplier MAR process captures self-reinforcing dependencies while a white noise process controls correlation structure among wavelet coefficients, which are then generated via a nonlinearity. The paper considers only fixed forms of nonlinearity and fix the order of the multiplier process making the underlying representation suffer in expressiveness [Kivinen et al., 2013a].

In Kivinen et al. [2007a,b] a hidden Markov tree of unbounded number of hidden states is built to learn global models for images, which appropriate dimensionality is automatically learned from the data using nonparametric Bayesian methods. Wavelet coefficient vectors are also in these HDP-HMT models marginally described using infinite mixtures of Gaussians, but they are generalized from GSMs to have discrete mixing distributions with the use of Dirichlet process priors. Fixed tree structures are traditionally linked with estimation errors in form of boundary artifacts. One potential way to reduce such estimation errors is increasing the state space [Willsky, 2002]. In the HDP-HMT dependencies between features are captured with an unbounded number of states, dimensionality of which is automatically inferred from the training data in an adaptive way[6].

In the models built in this work the features are *learned* from the data rather than being fixed as with wavelet transforms. The learned features turn out to have differences qualitatively when learning them for different kinds of data (such as different natural textures or natural textures compared to generic natural images; see and compare the results in Chapter 4 and Chapter 5, for example), or when training them according to a generative or a discriminative modelling application

[6]New states can be learned when new data is observed.

(as will be found in Chapter 5). There are also other issues with wavelet-based image modelling, as mentioned in Sec. 2.2.1.2.

2.2.1.4 Stochastic image grammar models based on recursive compositional principles

Most of the image models based on directed graphical models above have focused on modeling images with shallow architectures, which are limited in terms of representing data at multiple levels of abstraction. There are several recent models proposed in the literature that address this issue using hierarchical, dynamic representations. Fundamental work includes the dynamic tree models proposed in Storkey and Williams [2003] and Adams and Williams [2003], in which each of the nodes in a latent variable tree is allowed to choose its parent. Many of the more recent ones are based on stochastic image grammars built by recursively grouping image features into more complex ones by using compositional principles. These compositions based on the groupings, are often called as parts, each part defined recursively from its subparts, at all representation levels.

Jin and Geman [2006] propose a compositional system based on AND/OR graphs (in which the OR nodes allow to choose between multiple graph structures when modeling data), which is adapted for representing license plates of cars. The model is actually not learned from the data, but is hand crafted instead. Fidler and Leonardis [2007] develop a compositional system for objects in images, and an algorithm for learning its recursive grammar using bottom-up grouping of frequently occurring features into more complex ones, starting from wavelet responses of natural images. The system uses several thresholding operations in learning and inference, and does not define a proper generative model. The system is closely related to that of Zhu et al. [2008], which has a compositional structure with a top-down stage to fill in missing structures in their AND/OR graph based system. See also Zhu and Mumford [2006] for an extensive compositional system based on AND/OR graphs. One major goal in the transformation equivariant RBM model developed in the following chapter is to obtain a 'compositional' system based on DBNs, where bottom-up/top-down inference in the face of ambiguity or missing data falls out naturally based on a fully specified generative model.

2.2.2 Feed-forward Neural Networks

The thesis will use feed-forward neural networks in Chapter 5, in a supervised visual boundary prediction task, predicting an existence of a visual boundary at each point in a previously unobserved image. The networks, which are trained in a supervised manner, contain only deterministic nodes, and have logistic activation functions. In the case of single-layer networks, they are equivalent to logistic regression models.

A major advantage of models with deterministic nodes over stochastic ones [Neal, 1992] is the typically much more simpler learning (and inference which is deterministic) for the models. Supervised learning of feed-forward neural nets with deterministic nodes is typically performed by gradient descent methods, for which the partial derivatives of the objective function with respect to the full set of network parameters can be computed using the chain-rule of differentiation, the computation of which is formalized and efficiently implemented by the back-propagation algorithm [Rumelhart et al., 1986].

The practical use of the models involves considering several 'tricks of the trade', involving those related to parameter initialization, and hyper-parameter settings such as learning rates, through-out the course of learning. See Bengio [2012], and Bishop [1995] for in-depth discussions on these in a generic setting. For convolutional neural networks Simard et al. provides several useful strategies for tuning the models.

2.3 Learning and Inference Methods for Neural Networks

Here we will describe learning and inference methods for neural networks, with emphasis on stochastic neural networks, and restricted Boltzmann machines in particular. The next subsection describes the high-level picture of typical maximum-likelihood based learning for RBMs. The detailed methodologies, Monte Carlo integration, Markov chain Monte Carlo sampling, and iterative optimization methods are discussed in sections 2.3.2, 2.3.3, and 2.3.4, respectively.

2.3.1 Parameter estimation for RBMs

Maximum likelihood learning is a commonly used method for learning various probabilistic models. The idea in the method is to estimate parameters of the model so as to maximize the probability of the training data under the model. The log-likelihood of training data assuming N training cases, and conditional on model parameters θ for an RBM, can be written as follows:

$$L(\theta) = \log p\big(\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(N)} \mid \theta\big) = -N \log Z + \sum_{n=1}^{N} \log \left\{ \sum_{\mathbf{h}} \exp\left\{ -\mathrm{E}\big(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta\big) \right\} \right\}. \tag{2.34}$$

At an optimum of $L(\theta)$ the partial derivatives of the data log-likelihood with respect to the model parameters will be zero. As shown in Appendix A.1, the partial derivatives take the following form:

$$\frac{\partial L(\theta)}{\partial \theta} = \sum_{n=1}^{N} \left(\left\langle \frac{\partial \mathrm{E}(\mathbf{v}, \mathbf{h} \mid \theta)}{\partial \theta} \right\rangle_{p(\mathbf{v},\mathbf{h}|\theta)} - \left\langle \frac{\partial \mathrm{E}\big(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta\big)}{\partial \theta} \right\rangle_{p\left(\mathbf{h}|\mathbf{v}^{(n)},\theta\right)} \right). \tag{2.35}$$

While analytical expressions can be available for the expressions, the first one of these at least, which results from the partial derivative of the partition function with respect to the model parameters, is intractable to evaluate for most real-world problems, and thus exact maximum likelihood learning is not possible for them. Many of the commonly used learning methods, including those used in the thesis, will use approximations of maximum-likelihood learning. In all of cases considered, the intractable integrations are approximated by Markov chain Monte Carlo (MCMC) methods. In the following we will discuss main ingredients of these. For tricks of the trade, and practical recommendations, see Hinton [2010b], Bengio [2009, 2012], Bengio et al. [2013].

2.3.2 Monte Carlo Averaging

Assume we are interested in computing an expectation

$$\int_{\mathcal{X}} g(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} = \mathbb{E}_{p(\mathbf{x})}[g(\mathbf{x})] = \langle g(\mathbf{x}) \rangle_{p(\mathbf{x})} = c, \tag{2.36}$$

where c is a scalar, and g is some function of $\mathbf{x}$. Let $\{\mathbf{x}^{(s)}\}_{s=1}^{S}$ denote S i.i.d. samples from $p(\mathbf{x})$. The Monte Carlo estimate for c is then [Andrieu et al., 2003, MacKay, 1998, Neal, 1993]

$$\hat{c} = \frac{1}{S} \sum_{s=1}^{S} g(\mathbf{x}^{(s)}) \approx c. \tag{2.37}$$

Increasing the number of the i.i.d. samples, the Monte Carlo error decreases. Assuming we can draw samples from the distributions under which we take expectations in (2.35), we can then obtain a Monte Carlo estimate of the partial derivatives of the log-likelihood with respect to model parameters by averaging the partial derivatives of the energy functions with respect to the parameters evaluated at the samples. Sampling from the densities under which we take the expectations is non-trivial, but several methods are available which can be effective. In our case we will be using Markov chain Monte Carlo methods, which use specifically constructed Markov chains in the procedure.

2.3.3 Sampling from complicated high-dimensional distributions using Markov chain Monte Carlo methods

In the following, we will discuss shortly several MCMC methods. For a review on the methods for probabilistic inference, see Neal [1993]. Andrieu et al. [2003] provide an introduction to MCMC methods for machine learning.

2.3.3.1 Markov Chains

Markov chains can be seen as Bayesian networks with a chain structure. The joint probability density over the random variables factorizes as the marginal distribution of the initial node times the product of the probability distribution of each successor node conditional on its parent node.

In Markov chains, the initial state is chosen according to an initial state probability distribution, and the parent-conditional distributions are called transition probabilities. In stationary/homogenous Markov chains, the transition probabilities are the same at all positions, and define a transition probability kernel. The kernel parametrizes a Markov process, in which the probability of a state in some state-space at any specific time given all earlier states in the process, is dependent only on the previous state, and is determined according to the transition probability kernel value. Note that computing the unconditional probability of being in a particular state at a specific time will need to integrate over all of the possible states to arrive to the particular state. As an example, assuming a

discrete state space $\mathcal{X}$, we can write that

$$p(\mathbf{x^{(t)}}) = \sum_{\mathbf{x^{(t-1)}} \in \mathcal{X}} p(\mathbf{x^{(t)}}, \mathbf{x^{(t-1)}}) = \sum_{\mathbf{x^{(t-1)}} \in \mathcal{X}} p(\mathbf{x^{(t-1)}}) p\big(\mathbf{x^{(t)}} \mid \mathbf{x^{(t-1)}}\big) \quad (2.38)$$

$$= \sum_{\mathbf{x^{(t-1)}} \in \mathcal{X}} p(\mathbf{x^{(t-1)}}) \mathrm{T}(\mathbf{x^{(t)}}, \mathbf{x^{(t-1)}}), \quad (2.39)$$

where $p(\mathbf{x^{(t)}})$ denotes state probabilities at time t, T denotes the associated transition probability kernel matrix, and $\mathrm{T}(\mathbf{x^{(t)}}, \mathbf{x^{(t-1)}})$ denotes the transition probability from state $\mathbf{x^{(t-1)}}$ to $\mathbf{x^{(t)}}$, $p\big(\mathbf{x^{(t)}} \mid \mathbf{x^{(t-1)}}\big)$. Note that $p(\mathbf{x^{(t-1)}}) = \sum_{\mathbf{x^{(t-2)}}} p(\mathbf{x^{(t-1)}}, \mathbf{x^{(t-2)}})$ and thus the computation structure recurses until the initial time is reached, i.e. until the computation of $p(\mathbf{x^{(1)}}) = \sum_{\mathbf{x^{(0)}}} p(\mathbf{x^{(1)}}, \mathbf{x^{(0)}})$.

Then given any initial state in the state-space, repeated state sampling according to the transition probability kernel will simulate the Markov process, perfectly so at equilibrium, where the initial condition has been forgotten, and the state probabilities have converged (i.e. $p\big(\mathbf{x}^{(t)} \mid \mathbf{x}^{(0)}\big) = p\big(\mathbf{x}^{(t+1)} \mid \mathbf{x}^{(0)}\big) = \ldots = \lim_{t\to\infty} p\big(\mathbf{x}^{(t)} \mid \mathbf{x}^{(0)}\big) = \lim_{t\to\infty} p(\mathbf{x}^{(t)})$) and define the so-called equilibrium distribution.

2.3.3.2 Markov chain Monte Carlo

In Markov chain Monte Carlo (MCMC) methods, Markov chains are constructed so that the equilibrium distribution matches the distribution of interest. In our case this could be the probability distribution of the visible units an RBM defines. Drawing a sample from the distribution can be then done by initializing the state of the random variable, simulating the Markov chain dynamics (updating the states of the random variables conditional on the previous states), and recording the state of the chain after the initial condition has been forgotten. See e.g. Andrieu et al. [2003] on properties required for the transition probability kernel to define a valid MCMC method.

It is often important to pay attention to the conditional independence assumptions in the distribution of interest, when using MCMC methods. This is because they might suggest the specific form of the sampling scheme to use and/or not to use. Many samples can be collected simply by simulating multiple chains with independently chosen initial conditions, and recording a state as above. Many samples can be collected from a single converged chain, but one needs to take care in ensuring the samples are effectively i.i.d., and so for example neighboring states should not be used (as they are dependent by definition).

The commonly used MCMC algorithms are Metropolis-Hastings, Gibbs sampling, and Hybrid Monte Carlo, described in Sec. 2.3.3.3, 2.3.3.4. and 2.3.3.5 respectively.

2.3.3.3 Metropolis-Hastings algorithm

In Metropolis-Hastings (MH)-algorithms [Hastings, 1970], at each iteration new state $\mathbf{x}^*$ is proposed/sampled based on the current state $\mathbf{x}^{(t)}$ with an arbitrary user-defined state-transition proposal probability distribution q (i.e. $\mathbf{x}^* \sim q(\mathbf{x}^{(t)}, \mathbf{x}^*)$). The proposal $\mathbf{x}^*$ is then either accepted or rejected, with acceptance probability as

$$T(\mathbf{x}^{(t+1)} = \mathbf{x}^*, \mathbf{x}^{(t)}) = \min\left\{1, \frac{p(\mathbf{x}^*)q(\mathbf{x}^{(t)}, \mathbf{x}^*)}{p(\mathbf{x}^{(t)})q(\mathbf{x}^*, \mathbf{x}^{(t)})}\right\}. \tag{2.40}$$

If the proposal distribution is symmetric, the q-specific terms cancel out, and the approach is the same as that of the Metropolis algorithm [Metropolis et al., 1953] ($\min\left\{1, \frac{p(\mathbf{x}^*)}{p(\mathbf{x}^{(t)})}\right\}$). It is noteworthy that the distribution p needs to be able to be evaluated only up to a normalization constant, as it cancels out in the ratio of the p-specific terms. This is a very useful property for models such as Boltzmann machines in which the log-partition function is intractable to compute under non-trivial setups.

2.3.3.4 Gibbs sampling

The Gibbs-sampler [Geman and Geman, 1984] is a special case of the MH-algorithm. In the basic formulation, random variables are sequentially sampled, each from their full conditional distribution, which conditions on all of the other variables, at their latest states. The sampled values are the proposals under the MH, which however are always accepted. Often the practical applications concern models in which the full conditionals simplify onto distributions conditional only a relatively few random variables, and analytical expressions are available. The fact that there are no tunable parameters can make the approach appealing in practice.

The theory allows variables to be updated in blocks, and such blocked sampler might mix (explore the state-space) better than a non-blocked one. Assuming there are two sets of random variables $\mathbf{h} = \{h_1, \ldots, h_J\}$, $\mathbf{v} = \{v_1, \ldots, v_I\}$, initialized to values $\mathbf{h}^{(0)}$, and $\mathbf{v}^{(0)}$, respectively, the sampling alternates (until t below

is sufficiently large) as follows:

$$\vdots \tag{2.41}$$

$$\mathbf{h}^{(t)} \sim p\big(\mathbf{h} \mid \mathbf{v}^{(t)}\big) \tag{2.42}$$

$$\mathbf{v}^{(t+1)} \sim p\big(\mathbf{v} \mid \mathbf{h}^{(t)}\big) \tag{2.43}$$

$$\mathbf{h}^{(t+1)} \sim p\big(\mathbf{h} \mid \mathbf{v}^{(t+1)}\big) \tag{2.44}$$

$$\vdots \tag{2.45}$$

In RBMs with a bi-partite structure the hidden units are conditionally independent of each other given the visible units, and vice versa, and so the variables in the blocks can be sampled independent of each other. Note that if the ordering of the sampling with full conditionals is such that when sampling random variables in a set, they are updated all before the variables in the other set are updated again, the sampling algorithm will be the same as above. The thesis considers the ordering/blocking-setup with Gibbs sampling, which will be used within Chapters 3, 4 and 5 to draw samples from several RBMs. In other cases, which would be likely unwise, the sampling would be different, and would be expected to mix slower. Asymptotically any ordering and/or blocking (assuming every variable will be updated) would, however, result in the same results.

Problematic conditional distributions are those which include high correlations between the random variables, due to the fact that the chain moves in axes-aligned steps, and typically can be seen as performing a random walk in the state-space. As the correlations increase, increasingly more (and smaller) steps are expected to be needed to explore a specific distance along the major axes of them [Neal, 1993, Sec. 4.4].

2.3.3.5 Hybrid Monte Carlo

Hybrid Monte Carlo (HMC) [Duane et al., 1987] is a Metropolis-Hastings based MCMC method for models with continuous random variables which can be expressed using the energy-based notation (as say in eq. 2.3), and for which the partial derivatives of the associated energy function with respect to the random variables are computable. The method avoids the random walk behaviour described above, and provides potential means for making the exploration of the state-space much faster [Neal, 2011]. It can also provide computationally less demanding approach than Gibbs sampling for sampling from certain RBMs, by

integrating out the hidden units analytically; the thesis uses the method for sampling from PoT-based models in Chapter 4, from mcRBM models in Chapter 5, and from some GB-RBM-based models in Chapter 4 and in Appendix C.1.1.

The method adds auxiliary variables to the model, called momentum variables $\mathbf{m}$, one for each of the random variables in $\mathbf{x}$ (called position variables), and aims to produce samples from the following joint distribution:

$$p(\mathbf{x}, \mathbf{m}) = \frac{1}{Z} \exp\{-F(\mathbf{x}) - K(\mathbf{m})\}, \tag{2.46}$$

where $F(\mathbf{x})$ is the free-energy of the probability density of the marginal distribution of $\mathbf{x}$, and typically the kinetic energy $K(\mathbf{m}) = \frac{1}{2}\mathbf{m}^\top\mathbf{m}$. Together the random variables will be used to describe the state-space of a dynamical system simulated according to Hamiltonian dynamics, in which the sum of the energies remains constant. The dynamical update is typically implemented using the Leapfrog-algorithm, and due to discretization errors and non-perfect simulation occurring, the numerator and denominator will not exactly cancel out in the Metropolis-Hastings acceptance/rejectance criterion, and states will be rejected.

Each iteration of such algorithm involves choosing/sampling a random momentum, followed by the updating of the position and momentum variables by a number of Leapfrog-steps using some step-size, and accepting the updated position with probability according to the minimum of one and the ratio of the joint probabilities of the position and momentum variables, under the initial and updated states (i.e. $\min\left\{1, \exp\left\{F(\mathbf{x}^{(t)}) - F(\mathbf{x}^*) + K(\mathbf{m}^{(t)}) - K(\mathbf{m}^*)\right\}\right\}$). As the acceptance/rejectance-decision is based on the joint random variable configuration, low-acceptance probability configurations under the marginal model containing only the random variables of interest can become accepted with high-probability given suitable configurations of the momentum variables. This could mean for example escaping from a local optima. Furthermore, as the updates in the position take into account the partial derivatives of the energy-landscape of the model, random walk behaviour will be reduced. However, effective use of the method involves tuning several hyperparameters, which can be hard in practice. See Neal [2011] for more details.

See Zhang and Sutton [2011] for a recent advance to improve convergence speed/effectiveness of the method by utilizing approximate second-order gradient (Hessian) information about the target density from first-order information within an adaptive setting.

2.3.3.6 Practicalities

To obtain samples from the correct distribution, the MCMC-sampler needs to forget the initial configuration; the sampler needs to be 'burned-in'. Although methods exist for the estimation of the burn-in times (such as the potential scale reduction factor [Brooks and Gelman, 1998]), this estimation can be complicated in practice. In many machine learning applications, this is not feasible in practice, and computational requirements often lead to relatively early sample collection (certainly so with respect to asymptotic conditions, where guarantees of consistency are strong). Several i.i.d. samples can be obtained by running as many chains as samples that are wanted, initialized to different randomly chosen initial conditions, and setting the samples to the last states of the chains after burn-in. If more samples are wanted than the number of chains run, several states will need to be recorded from the chains. The selection of the states onto samples will need to take into account that neighboring states within the chains are dependent. Taking them by offsets, as in the method of thinning (see for example Gelman et al. [2003]) will reduce the dependence of the samples. The thesis uses Constrastive Divergence-based methods described in 2.3.4.2 for learning Boltzmann machines, and here thinning is actually not necessary. For the synthesis task, however, this is a valid consideration.

In synthesis the thesis considers multiple chains (as many as samples are needed), and from each collects only the last state. In the constrained synthesis tasks of inpainting in Chapter 4 and in Appendix C.1.1, the inpainting frame specifies which regions to be inpainted, and these values are initialized to zeros. In unconstrained synthesis the thesis uses full-zero images and/or samples from a spherical-covariance and zero-mean Gaussian. We run the chains in a parallel fashion on a GPU, and at any particular iteration the chains use different random number generator states. The number of iterations were set between 5000-100000 iterations, chosen based on three necessary criteria to be met: (i) computational feasibility, (ii) compatibility for comparisons, and (iii) refuting of non-convergence by visual monitoring of samples and/or monitoring of their free-energies.

When using HMC, we used either 20 or 30 Leapfrog steps. During learning the associated step-size was adapted at to obtain and maintain (around) 0.9 acceptance rate, but held constant for sampling from a particular model instantiation. During inference the step-size was fixed after a short adaptation period during

which it was set (by trial and error adjustment of associated parameters) to obtain (around) 0.65 acceptance rate once fixed. The adaptation produced stochasticity onto the final fixed step-sizes across different runs/starts. The adaptation measured the acceptance rate as an exponentially weighted moving average of observed values with a smoothing factor of 0.9. The step size was increased or decreased by an exponential factor similar to the bold-driver technique (as described in Bishop [1995, Sec. 7.5.3.]). This factor was however adaptive and dependent on whether it was previously changed towards the same direction (increase or decrease), changed also by using an exponential factor. The step-sizes were also set minimum and maximum accepted values, and the initial step-size was set to relatively small value compared to the final fixed one.

Appendix C.1.1 and Sec. 4.4 considers both the above sampling technique and block-Gibbs sampling (which does not have tunable parameters) for unconstrained and constrained synthesis from several diagonally-tiled-convolutional RBM-based models, for performance texture synthesis and inpainting, respectively. Performance metrics under the tasks with the approaches were similar, and so were the samples qualitatively.

2.3.4 Iterative optimization methods

This subsection discusses iterative optimization methods for parameter estimation, with focus on stochastic gradient descent methods, which will be used in learning all of the models developed in the thesis. In the following the basic gradient descent algorithm will be discussed. Then we will describe in Sec. 2.3.4.2 Contrastive Divergence-based methods, which use gradient descent methods in learning Boltzmann-machines. Finally we discuss typical regularization methods used within the learning; encouraging sparse feature activations in Sec. 2.3.4.3 and parameter decay in Sec. 2.3.4.4.

2.3.4.1 Gradient descent methods

Assume an objective function $L(\theta)$ defined in the space of variables θ. Suppose the task is to find a variable configuration which corresponds to an optimum under the objective function. In our case the variables θ would correspond to model/network parameters, and the function L would be for example the negative log-likelihood of the data. At each optimum (local or global), the partial derivatives of the

function with respect to the parameters will need to equate to zero. In practice, solving the associated equation for the parameters of the model is often not possible, and iterative methods are resorted to. This is the (necessary) scenario for training all of the models considered in the thesis.

The methods start from some initial configuration of the variables $\theta^{(0)}$, and iteratively update their configurations as follows:

$$\theta^{(t)} \leftarrow \theta^{(t-1)} + (\Delta\theta)^{(t)}, \tag{2.47}$$

where $(\Delta\theta)^{(t)}$ defines the position increment, a step, which has a specific direction, and a size/length along that direction.

Gradient descent updates the configuration by moving it towards the direction of the negative of the partial derivative of the objective function with respect to the variables:

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \alpha^{(t)} \nabla L(\theta), \tag{2.48}$$

where $\alpha^{(t)}$ denotes a non-negative constant known as the learning rate. As can be seen from the equation, the size/length of the step that is made, is controlled by the gradient value, and the learning rate. In a batch-version of the gradient descent, all of the data points are used in the computation of $\nabla L(\theta)$. Provided the step-size is sufficiently small, the approach is guaranteed to improve the configuration, unless it it zero due to the gradient being zero, in which case the configuration defines an optimum.

Often it is not possible in practice to consider all of the data points to update the configurations, and better performance can be obtained (for example in a computational resources limited setting) in a sequential/stochastic-version of gradient descent, in which only a subset of the data is used in the evaluation of the gradient at each step. Typically the subsets are equally-sized sets of the training data, called mini-batches [Bengio, 2012].

The initialization, and the selection or the scheme how to set up the learning rates, can be crucial in practice. There exist techniques for optimizing the step-size selection such as line-search [Bishop, 1995] (where the step-size is selected so that the updated position optimizes the objective function values along the line from the current point to the direction of the gradient sufficiently far away), but for the problems considered in this work, they are not in general feasible and/or useful.

In many cases the gradient descent might produce oscillating updates. One approach to reduce oscillations is via the reduction of the learning rate(s). Another is by using a 'momentum'-term in the configuration update [Bishop, 1995], so that

$$(\Delta\theta)^{(t)} = -\alpha^{(t)}\nabla L(\theta) + \beta^{(t)}\,(\Delta\theta)^{(t-1)}\,, \tag{2.49}$$

where $\beta^{(t)} \in (0, 1)$, and defines the rate of the momentum for the epoch. The use of momentum is often adopted in learning RBMs with the Contrastive divergence learning approach described below. As for example Hinton [2010b] points out, the parameters no longer change according to the direction of the steepest descent. However, learning can nevertheless be faster [Hinton, 2010b, Bishop, 1995].

If the objective function and its partial derivatives can be computed exactly, which in our case is typically not the case, methods using local second-order derivative information about the optimization landscape, such as the conjugate gradient algorithm are available, and might provide more effective optimizers. See for example Bishop [1995] for more details on such algorithms for the class of models, and also on further properties on the gradient descent. For the latter, see also [Bengio, 2012] and the suggested deepening reference [Bottou, 2012] therein.

2.3.4.2 Constrastive divergence-based algorithms

Contrastive divergence (CD) [Hinton, 2002b] is a method commonly used for learning restricted Boltzmann machines, and many other energy-based models [Teh et al., 2003]. The approach starts from a gradient descent scenario with gradient defined as in (2.35). As previously mentioned, computing the exact gradient under an RBM with more than few hidden units intractable. In computing the gradient, the CD method replaces the distribution over which the first expectation is taken in (2.35), the model distribution, by a different distribution. The averaging distribution in the second term is left intact, but as with the first term, the expectation will be approximated by using Monte Carlo averaging. Samples for the second term, called 'positive phase' data, can be obtained directly. Samples for the first term, called 'negative phase' data, are obtained by MCMC.

In CD-k, the Markov chains are initialized to the training data, and instead of running sampling until the equilibrium distribution is reached, only one or a few steps (k) will be taken. This will reduce variance in the resulting estima-

tors, but introduces bias; the approach has been shown to work well in several applications [Hinton, 2010a, Bengio, 2009].

In the persistent chains CD [Tieleman, 2008, Zhu et al., 1998, Younes, 1988], the data for the negative phase is given by a maintained set of negative particles. The negative particles are in the beginning of the algorithm started from some setting, such as training data, or set some other way such as randomly. After the initial condition, no further reference to the training data will be made. The negative particles are updated with MCMC with the current model distribution as the distribution with sampling of interest.

Outside of the setting of the negative phase data, the algorithm is the same as the CD. However, the approach can be made to perform maximum likelihood estimation asymptotically, as opposed to the basic CD (assuming finite k). Although learning is typically slower than with the basic CD (due to the fact that learning rates need to be typically lower to not make the learning diverge), the approach is considered to be preferable for density modelling [Hinton, 2010b]. The algorithm was proposed for RBMs in Tieleman [2008], but the same estimation idea has been used in different but still MRF-based contexts significantly earlier, for example in Zhu et al. [1998], and in Younes [1988].

The fast-persistent chains CD [Tieleman and Hinton, 2009] considers two sets of parameters, so-called regular parameters, and fast parameters, the sum of which is used to describe the parameters for the model distribution. In practice the fast parameters are associated with a larger learning rate than the regular ones, but the fast parameters (not necessarily their learning rates) are annealed during the course of learning, and will vanish asymptotically. The scheme is demonstrated to increase mixing from the PCD. See Bengio et al. [2013] for a more in-depth discussion on the methods, and other commonly used learning techniques for the class of models considered. See Hinton [2010b] for practical recommendations in the context of RBMs.

2.3.4.3 Regularization: Encouraging sparse feature activations

Encouraging sparse feature activations is common regularization method in unsupervised feature learning. While there are biological motivations using sparsity [Olshausen and Field, 1996, 1997], typically the motivation for having such a term in the learning objective function relates to having interpretability in the features. Discriminative settings may also benefit from it [Hinton, 2010b].

A commonly used objective function for model learning is of the following form:

$$C(\theta) = C_1(\theta) - \lambda C_2(\theta), \tag{2.50}$$

where $C_1(\theta) = -\frac{1}{N}\sum_{n=1}^{N} \log p(\mathbf{v}^{(n)}|\theta)$ is the negative log-likelihood of data, acting as a data-fit term, and the second term is the term for encouraging sparsity, where λ is a regularization constant. A commonly used form for $C_2(\theta)$ under an RBM is the following:

$$C_2(\theta) = \frac{1}{J}\sum_{j=1}^{J} \mathcal{H}\left(p^*, q_j\right) \tag{2.51}$$

where $\mathcal{H}\left(p^*, q_j\right)$ is the cross-entropy between a Bernoulli target distribution p^* with success probability a, and the distribution $q_j = \frac{1}{N}\sum_{n=1}^{N} p\big(h_j \mid \mathbf{v}^{(n)}, \theta\big)$ (which records the average probability of the unit h_j being off or on over a dataset). The goal of the sparsity term (as used in Lee et al. [2008], Nair and Hinton [2009]) is to encourage a hidden unit to come on a proportion a of the time; typically a is set to a small value close to zero. A closely related form is the following:

$$C_2(\theta) = \frac{1}{N}\sum_{n=1}^{N} \mathcal{H}\left(p^*, q^{(n)}\right), \tag{2.52}$$

where $q^{(n)} = \frac{1}{J}\sum_{j=1}^{J} p\big(h_j \mid \mathbf{v}^{(n)}, \theta\big)$. Notice that this approach encourages the average hidden unit activations per image to be a, rather than the average of activations of a feature over a set of images. In the RBM training guide [Hinton, 2010b], Hinton recommends to set the distribution q according to an exponentially weighted average (rather than directly based on the observed activation level; this assumes online learning setting based on the use of mini-batches). This scheme is also adopted in [Nair and Hinton, 2009]. In the approach of Ngiam et al. [2011] both the sparsity of a feature over different examples (lifetime sparsity), and that of the features together to a specific example (population sparsity) is encouraged. None of these approaches guarantee spatial coherence in the feature activations. One way to encourage such is to connect the hidden units laterally within their feature planes, possibly in a local way.

2.3.4.4 Regularization: Parameter decay

The commonly used regularization functions for the parameter decay are the following [Hinton, 2010b]:

$$\mathrm{L}_2(\theta) = \left(\sum_i \theta_i^2\right)^{1/2} \tag{2.53}$$

$$\mathrm{L}_1(\theta) = \sum_i |\theta_i|, \tag{2.54}$$

where θ denotes the parameters for the regularization. When used in learning, the overall objective function includes the regularization function multiplied with a regularization constant (say λ) hyperparameter.

There are two main motivations for using parameter decay in learning: one is to make the learning more robust, including to avoid overfitting and the divergence of learning. The other one is to obtain more interpretable parameters.

Chapter 3

Transformation Equivariant Boltzmann Machines

3.1 Introduction

We consider the problem of using deep belief net (DBN) architectures to model the structure occurring in natural images. One of the desiderata for a computer vision system is that if the input image is transformed (e.g. by a translation of two pixels left), then the inferences made by the system should co-transform in a stable, and predictable way; this is termed *equivariance* (see discussion on intensity and translation equivariance in Nair and Hinton [2010b]). This behavior has been motivational in the development of steerable filters [Simoncelli et al., 1992] (where it is termed shiftability), and we argue that obtaining such transformation equivariant representations is important for the architectures that we are considering as well. *Translational* equivariance is readily built in by a convolutional architecture as found in neural networks [Waibel et al., 1989, LeCun et al., 1998], and more recently for RBMs, see e.g. Lee et al. [2009]. However, there are additional transformations that should be taken into account: in this work we focus on equivariance with respect to in-plane *rotations*. Building in such property is important to avoid the system having to learn rotated versions of the same patterns at all levels in the network. For example in Fig. 2 of Lee et al. [2009] many of the learned filters/ filter combinations shown are rotated versions of each other. The goal of this work is to build a DBN architecture that is translation and rotation equivariant. To do this we introduce a novel kind of rotational/steerable unit for Boltzmann machines, as described in section 3.2. Section 3.3 discusses

inference and learning for the models, and Section 3.4 conducts experiments with them using artificial and real image data.

One of the inspirations for this chapter is the work of Fidler and Leonardis [2007], in which conjunctions of edge and bar (sine and cosine) Gabor features are built up into more complex patterns that occur frequently in the input image ensemble. Their architecture is translation and rotation invariant. However, their method does not define a generative model of images, but rather performs a layerwise grouping of features from layer $\ell - 1$ to create features at layer ℓ. This means that it is heavily dependent on various thresholds used in the learning algorithm, and also that it is unable to carry out bottom-up/top-down inference in the face of ambiguous input or missing data. We show how such translation and rotation invariant groupings arise naturally in a fully-specified multi-layer generative model.

3.2 Building in Transformation Equivariance

We first discuss the rotation-equivariant restricted Boltzmann machine (STEER-RBM) model which has one hidden layer; this hidden layer contains the 'steerable' units which are a particular feature of our architecture. Next in section 3.2.2 we describe a translation equivariant version of the model, and finally in section 3.2.3 generalize this to a deep belief net, which is the multi-hidden-layer generalization of the translation and rotation equivariant model.

3.2.1 Rotation Equivariant RBMs

The key feature of the STEER-RBM is the construction of the stochastic steerable hidden units, each of which combines a binary-valued activation variable h_j turning the unit on/off with an associated discrete-valued rotation variable r_j taking on possible states $k = 1, \ldots K$, whose effect is to in-plane rotate the weights of the unit by $360(k-1)/K$ degrees. Let $W_j(\cdot, 1)$ be the canonical pattern of weights connecting hidden unit h_j to visible units $\mathbf{v}$ under no rotation. The transformed weights $W_j(\cdot, k)$ for rotation k are derived from the canonical view using geometrical knowledge of in-plane rotations, so that

$$W_j(\cdot, k) = \mathbf{R}^{(k)} W_j(\cdot, 1) \quad \Rightarrow \quad W_j(i, k) = \sum_{\ell=1}^{D} R^{(k)}(i, \ell) W_j(\ell, 1), \tag{3.1}$$

where $\mathbf{R}^{(k)}$ is a *fixed* $D \times D$ transformation matrix applying an in-plane rotation of $360(k-1)/K$ degrees, and D denotes the number of pixels/visible units in $\mathbf{v}$. Note that by choosing K large we can approximate rotations to any desired accuracy. An example of this rotation in action is shown in the top row of Figure 3.3. In our implementation, we bilinearly interpolate the weights into their new locations, such that each of the elements in the rotated view is computed as a convex combination of (maximally) four neighboring rotated canonical weights, each of which have been rotated about the center of the canonical weights plane[1]. Thus each row of the rotation matrices maximally contains four non-zero elements which sum to one.

Given this architecture, the joint probability density of a STEER-RBM model consisting of visible units $\mathbf{v}$ and binary hidden units $(\mathbf{h}, \mathbf{r})$ is given by the Boltzmann distribution $p(\mathbf{v}, \mathbf{h}, \mathbf{r} \mid \theta) \propto \exp\{-\mathrm{E}(\mathbf{v}, \mathbf{h}, \mathbf{r} \mid \theta)\}$ with the following energy, assuming continuous, conditionally Gaussian units:[2]

$$\mathrm{E}(\mathbf{v}, \mathbf{h}, \mathbf{r} \mid \theta) = \sum_i \frac{v_i^2 - 2av_i}{2\sigma^2} - \sum_j h_j b_j - \frac{1}{\sigma} \sum_j h_j \sum_i v_i W_j(i, r_j) \quad (3.2)$$

where $\theta = \{\mathbf{a}, \mathbf{b}, \mathbf{W}, \sigma\}$ consist of hidden unit biases $\mathbf{b}$, visible unit biases $\mathbf{a}$, connection weights $\mathbf{W}$, and the standard deviation of the Gaussian conditional distributions of the visible units σ. The energy function for binary visible units can be obtained by removing the quadratic term v_i^2, and setting σ to unity. As $W_j(i, r_j) = \sum_{k=1}^{K} \delta(k, r_j) W_j(i, k)$, the model defines a mixture of RBMs, but in contrast with the implicit mixture RBM model of Nair and Hinton [2009], there is parameter sharing between the mixture components due to rotation equivariance. Although we have described RBMs above, extensions of other energy-based models to use rotational units could be also considered, such as conditionally full-covariance Gaussian models [Ranzato et al., 2010b].

3.2.2 Rotation and Translation Equivariant RBMs

To learn models for whole images, a translation equivariant extension of the STEER-RBM is used, assuming a reduced connectivity structure so that a hidden

[1] To avoid boundary artifacts with non-circular receptive fields, one can zero pad the canonical weights plane such that each of the rotated canonical weights are within the boundaries defined by the extended plane for any rotation angle.

[2] The joint probability density of the visible units conditional on the hidden units and model parameters factorizes as a multivariate spherical-covariance Gaussian.

unit h_j is connected to a subset of visible units specified by a receptive field system, and parameter sharing is used so that the responses of units to a stimulus are translation equivariant. We call the hidden units sharing these parameters a *feature plane*. To extend convolutional RBMs, the STEER-RBM also adds input rotation equivariance to hidden unit activation. Thus we consider a weight kernel ω_α for feature plane α, which is sufficient to define the connection weights between the hidden units in feature layer α and the visible units. The energy function for the convolutional STEER-RBM is then

$$\mathrm{E}(\mathbf{v}, \mathbf{h}, \mathbf{r} \mid \theta) = \sum_i \frac{v_i^2 - 2av_i}{2\sigma^2} - \sum_{\alpha,j} h_{\alpha j} \left(b_\alpha + \frac{1}{\sigma} \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \, \omega_\alpha(d(j,\ell), r_{\alpha j}) \right) \quad (3.3)$$

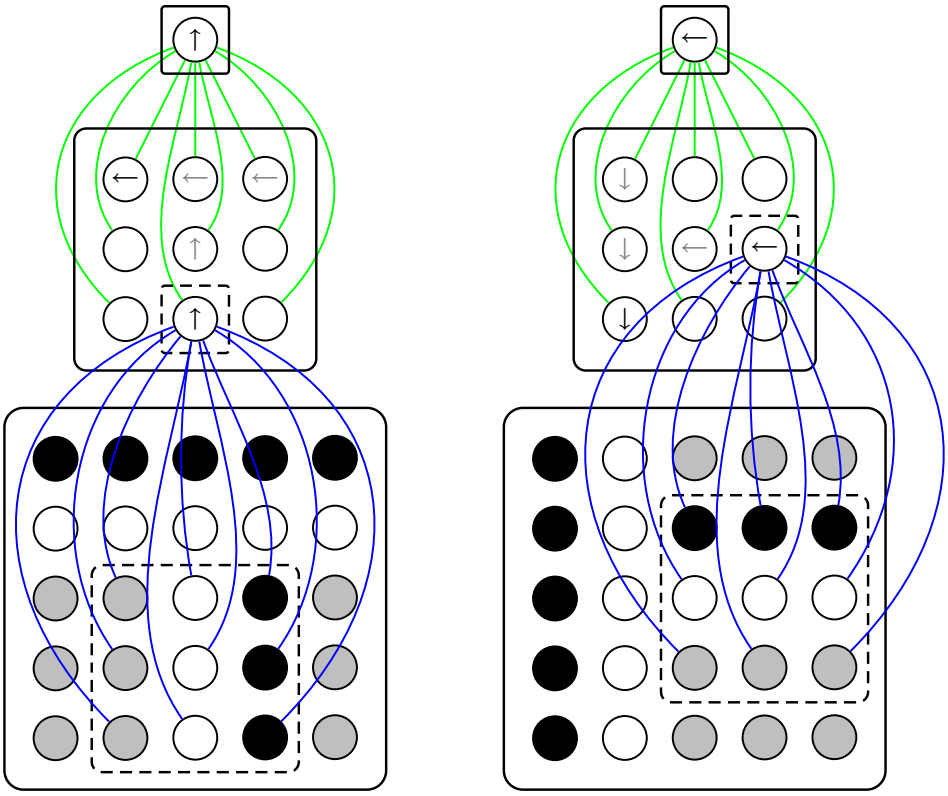
where a is visible unit layer bias, b_α is the bias for hidden unit feature plane α, $\mathrm{N}_{\alpha j}$ indexes the visible units within the receptive field of hidden unit $h_{\alpha j}$, σ defines the standard deviation of the univariate Gaussian conditional distribution $p(v_i \mid \mathbf{h}, \mathbf{r}, \theta)$, and $d(j,i)$ computes the spatial-offset dependent index of the weight kernel weight that is used to connect hidden unit h_j to v_i.

3.2.3 Rotation and Translation Equivariant Deep Belief Nets

To learn higher-level patterns from images, we follow the DBN approach of Hinton et al. [2006], stacking multiple layers of convolutional STEER-RBMs on top of each other. In this model, each of the hidden units in a higher level STEER-RBM is connected to a subset of the hidden units in each of the feature planes in the hidden layer below, again via by a receptive field system. As both the higher and lower level units are rotational, we now need a *triply* indexed weight parameter $\omega^{\ell_\alpha}_{\ell-1_\beta}(j,m,k)$ which connects a unit in feature plane α in layer ℓ to feature plane β in layer $\ell-1$ below. Here j denotes the spatial offset, while m and k index the rotational states in the lower and higher layers respectively. Thus the energy function between layers ℓ and $\ell-1$ is of the following form:

$$\begin{aligned} \mathrm{E}\big(\mathbf{h}^{\ell-1}, \mathbf{r}^{\ell-1}, \mathbf{h}^{\ell}, \mathbf{r}^{\ell} \mid \theta^{\ell}\big) = &- \sum_\beta b^{\ell-1}_\beta \sum_i h^{\ell-1}_{\beta i} - \sum_\alpha b^{\ell}_\alpha \sum_j h^{\ell}_{\alpha j} \\ &- \sum_\alpha \sum_j h^{\ell}_{\alpha j} \sum_\beta \sum_{i \in \mathrm{N}^{\ell}_{\alpha j}} h^{\ell-1}_{\beta i} \omega^{\ell_\alpha}_{\ell-1_\beta}(d(j,i), r^{\ell-1}_{\beta i}, r^{\ell}_{\alpha j}). \end{aligned} \quad (3.4)$$

The computation of the transformed weights for these higher hidden layers has to be different from that of the first layer, since changing the rotational state of



(a) convolutional Steer-DBN patterns for a stimulus

(b) convolutional Steer-DBN patterns for a rotated stimulus

Figure 3.1: Illustration of how a Steer-DBN is constructed to behave under a pattern rotation.

a higher level pattern needs to rotate the lower level rotational states/patterns accordingly. See Figure 3.1 for an illustration.

The transformations for each feature can be again done by knowledge using fixed transformation operators, by first in-plane rotating the lower-level rotation-specific canonical weight matrix slices, and then circularly shifting the dimensions of the resulting matrix. The non-canonical view of a level ℓ weight kernel can be thus written as follows:

$$\omega^{\ell_\alpha}_{\ell-1_\beta}(j,m,k) \quad = \quad \sum_{\rho=1}^{K} \mathbf{S}^{(k)}(\rho,m) \sum_{\delta} R^{(k)}(j,\delta)\omega^{\ell_\alpha}_{\ell-1_\beta}(\delta,\rho,1), \tag{3.5}$$

where $\mathbf{S}^{(k)}$ is a *fixed* $K \times K$ binary matrix applying a circular shift (of $k-1$ shifts) forward to the columns of $\mathbf{R}^{(k)}\omega^{\ell_\alpha}_{\ell-1_\beta}(\cdot,\cdot,1)$.

3.3 Inference and Learning in the Models

As with standard RBMs, the conditional distributions of the hidden units are independent given $\mathbf{v}$ for the convolutional STEER-RBM. Thus we have for (3.3) that $p(\mathbf{h}, \mathbf{r} \mid \mathbf{v}, \theta) = \prod_\alpha \prod_j p(h_{\alpha j} \mid \mathbf{v}, \theta)\, p(r_{\alpha j} \mid h_{\alpha j}, \mathbf{v}, \theta)$, where

$$p(h_{\alpha j} \mid \mathbf{v}, \theta) = \frac{\sum_{k=1}^{K} \exp\left\{ h_{\alpha j} \left(b_\alpha + \frac{1}{\sigma} \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega_\alpha(d(j,\ell), k) \right) \right\}}{K + \sum_{k=1}^{K} \exp\left\{ b_\alpha + \frac{1}{\sigma} \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega_\alpha(d(j,\ell), k) \right\}} \tag{3.6}$$

$$p(r_{\alpha j} \mid h_{\alpha j} = 1, \mathbf{v}, \theta) = \frac{\exp\left\{ b_\alpha + \frac{1}{\sigma} \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega_\alpha(d(j,\ell), r_{\alpha j}) \right\}}{\sum_{k=1}^{K} \exp\left\{ b_\alpha + \frac{1}{\sigma} \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega_\alpha(d(j,\ell), k) \right\}}. \tag{3.7}$$

The key quantity in this computation is $a_{\alpha j}(k) = \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \; \omega_\alpha(d(j,\ell), k)$ which computes the dot product of the visible variables in $\mathrm{N}_{\alpha j}$ with weight kernel w_α at rotation k. (3.6) evaluates a nonlinear combination of these quantities summed over k compared to K in order to compute $p(h_{\alpha j} \mid \mathbf{v}, \theta)$. Similarly in (3.7) $p(r_{\alpha j} \mid h_{\alpha j} = 1, \mathbf{v}, \theta)$ is computed based on the relative strengths of the $a_{\alpha j}(k)$ terms. For a multi-layer network a crude approximation to full inference is to sample from the learned STEER-RBMs layerwise from bottom to top. More sophisticated alternatives are possible, such as the up-down algorithm described in Hinton et al. [2006], or e.g. some other Markov chain Monte Carlo sampling methods.

As usual with DBNs we learn the parameters of the models layer-wise. We have used stochastic gradient-descent based methods to train the models in the experiments, optimizing an objective function consisting of a data fit term, plus a term that encourages sparsity, according to equations (2.50) and (2.52)[3]. For a datafit term based on the log likelihood L, the gradient wrt a parameter θ is given by $\frac{\partial L}{\partial \theta} = \langle \frac{\partial E}{\partial \theta} \rangle^+ - \langle \frac{\partial E}{\partial \theta} \rangle^-$, where $\langle \cdot \rangle^+$ denotes expectation with the training data clamped, and $\langle \cdot \rangle^-$ the unclamped phase. In fact we generally use the contrastive divergence CD-1 approximation, described in Section 2.3.4.2, to the negative phase. To understand how the model learns under optimization, it is instructive to consider the partial derivatives of the energy function with respect to the canonical features. Assuming the model of (3.3), we have that

$$\frac{\partial \mathrm{E}(\mathbf{v}, \mathbf{h}, \mathbf{r} \mid \theta)}{\partial \omega_\alpha(\delta, 1)} = -\frac{1}{\sigma} \sum_j h_{\alpha j} \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \; R^{(r_{\alpha j})}(d(j,\ell), \delta). \tag{3.8}$$

[3]Non-sparsity is penalized proportional to a sum of feature-plane specific cross-entropies, each between a Bernoulli target distribution, and the distribution recording the average probability of a unit being off or on at the plane, similar to Nair and Hinton [2010a].

This has the effect of multiplying the visible pattern in $\mathrm{N}_{\alpha j}$ by $(\mathbf{R}^{(r_{\alpha j})})^T$. As this is a close approximation to applying a reverse rotation, patterns which are detected to be present in a non-canonical orientation, are rotated 'back' into the canonical view, in which the feature-specific canonical statistics are then updated. The learning is similar for the higher layer models, where the alignment also takes into account the lower unit's rotation assignment. Partial derivatives with respect to the biases take the standard forms.

3.4 Experiments

The following subsection experiments with a rotation equivariant RBM on modelling small patches of binary data. We then consider in Section 3.4.2 experiments on modelling natural image data, using also translation equivariant feature sharing, in a shallow architecture and in a deeper architecture.

3.4.1 Rotated Handwritten Letters

This experiment considered the problem of modeling data consisting of rotated versions[4] of handwritten letters and blank images, with STEER-RBM having a single hidden unit. Although using biases would help in this task because there exist locations within the images that are biased to be on or off, they were not used in order to demonstrate that it is possible to learn a good model even without them. See Figure 3.2(a) for a sample of training data examples. Learning was based on minimizing negative log-likelihood of the training data, where

$$\log p\big(\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(N)} \mid \mathbf{W}\big) = -N \log Z + \sum_{n=1}^{N} \sum_{j} \log g_j(\mathbf{v}^{(n)} | \mathbf{W}_j), \tag{3.9}$$

where

$$g_j(\mathbf{v}^{(n)} | \mathbf{W}_j) = K + \sum_{k=1}^{K} \exp\left\{ [\mathbf{v}^{(n)}]^\top \mathbf{R}^{(k)} W_j(\cdot, 1) \right\},$$

and the partition function

$$Z = \sum_{\mathbf{h},\mathbf{z}} \prod_{i} \left(1 + \exp\left\{ \sum_{j} h_j \left[\sum_{k=1}^{K} z_j(k) R^{(k)}(i, \cdot) \right] W_j(\cdot, 1) \right\} \right).$$

[4]Using 16 different rotation angles from the full range of possible rotations.

(a) Training data instances.

(b) Most likely visibles conditional on an active unit with the (16) different rotational states under learnt model.

Figure 3.2: Learning a STEER-RBM on a data set consisting of blank images and images of rotated handwritten letters.

A scaled conjugate gradient algorithm was used for the optimization [5]. Computation of the exact gradient was possible because the model contained only one hidden unit. The partial derivatives of the data log-likelihood with respect to model parameters (needed in the SCG-algorithm) are given in Appendix B.3. Figure 3.2(b) shows expected visible units conditional on an active unit with the different rotational states under learned model from 1000 training images. We can see from the figure, that the generative model *correctly* believes that the data consists of rotated 'E'-letters (and blank images - to save space the expected visibles conditional on turning off the hidden unit have been omitted).

3.4.2 Natural Image Data

We first learnt RBM models (3.3) from a set of whitened natural images [Olshausen and Field, 1996] using CD-1 learning[6]. Fig. 3.3 (left, top) shows the type of feature consistently learnt as the most significant, at various rotations. This is an "edge detector", similar to the features found e.g. in Lee et al. [2009] at various orientations. The bottom row shows a natural image patch, and most likely states colour-coded according to orientation, at each location. The responses occur at edge-like structures; notice the steady rotational response change, e.g. while tracing the outline of the central object. We have also trained this model with several feature planes; results using three are shown (right).

To validate the higher-layer learning we first considered modeling artificial

[5] The scg-function of the Netlab-toolbox version 3.3 was used.

[6] σ was set close to the data standard deviation. The total target activation for sparsity encouragement was 0.1.

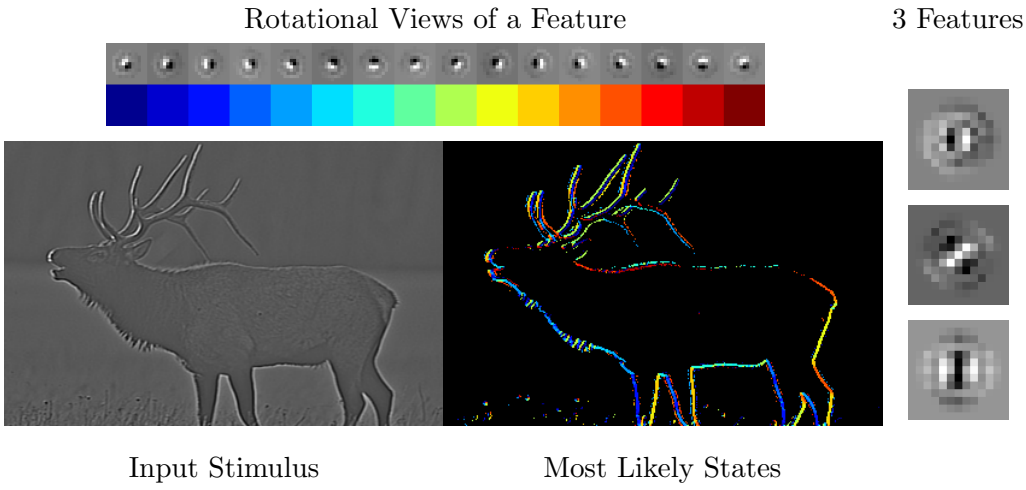
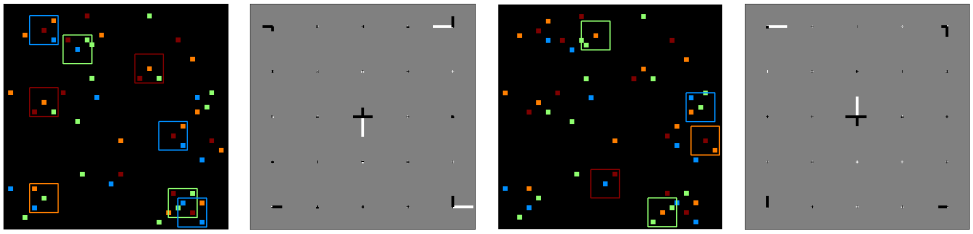


Figure 3.3: Left-top: Learned feature at various orientiations (with receptive field diameter = 9). Left-bottom: Whitened natural image region, and most likely unit states (colour-coded according to rotation). Right: Weights of the learned set of 3 features.

rotational pattern data simulating first-layer responses. Fig. 3.4 shows input patterns; the four colours denote four different orientation responses. There are two patterns in the noisy data, one consisting of 3 active inputs, and the other of 2. These are successfully learned, see panels 2 and 4, and the caption for more details. The higher-layer feature visualization extends the Hinton-plot from 2 to 3 dimensions: A Hinton-plot encodes a 2D-matrix of values with a grid/lattice of squares, with a square per matrix element, with colour encoding the sign, and the size encoding the magnitude of the respective element. Such could be used to visualize the first layer basis functions, with the squares encoding the basis function elements. In the higher-layer filters, there is an additional dimension which encodes the orientation of the lower-layer basis function from its canonical view. To visualize the higher-layer filters, we consider the Hinton-plot but replace each square with an oriented line-segment, with colour again endoding the filter element's sign, the length encoding the magnitude, and the orientation encoding the rotation of the lower-layer basis function from its canonical view under the particular position.

We have also applied the learning to the natural images, using the single edge-like feature in the first layer. The results (see Fig. 3.6 which shows the first-layer basis function and the second-layer basis functions, in their canonical views) show this yields higher-order conjunctions of this feature, such as extended and



Activations (Feat. 1) Feature 1 Activations (Feat. 2) Feature 2

Figure 3.4: Features learned from rotation colour-coded (→,↑,←,↓) artificial data containing rotated, randomly placed instances of two rotational shapes, in clutter. Panels 1 and 3 show the data, with the respective higher level features denoted by bounding boxes of the units' receptive field size centered on the unit location, and coloured according to the rotation assignment. In panels 2 and 4 the 5×5 canonical weight kernels are visualized using oriented black/white line segments, placed to start from an evenly spaced grid. The grid locations denote the spatial offsets for the weight kernels weights, the different orientations index the lower-level rotational states, the segment lengths denote the weight magnitude, and colour denoting the sign with black denoting a negative, and white denoting positive a weight. (Essentially this extends the Hinton-plot to deal with (multi-way-)oriented weights.)

curved edges, and intersections. Note that these features are similar to SIFT-descriptors [Lowe, 2004], but in a generative framework. Note also that only the canonical views of the features are shown in Fig. 3.6; see Figure 3.5 for an illustration of several views per a second layer feature.

3.5 Related Work

We have discussed above the work of Fidler and Leonardis [2007]. The work of Zhu et al. [2008] is similar to it, except that there is a top-down stage to the learning process (but not in the given inference algorithm) to fill in missing parts of the hierarchy. Both papers use hand-crafted algorithms for detecting groupings of lower-level features, involving various thresholds. In contrast we formulate the problem as a standard DBN learning algorithm, but build in transformation equivariance. One advantage of the DBN is that it is naturally set up for bottom-up/top-down inference in the face of ambiguity or missing data.

The orientation-adapted Gaussian scale mixture (OAGSM) model [Hammond and Simoncelli, 2008] describes how a Gaussian model for a wavelet coefficient

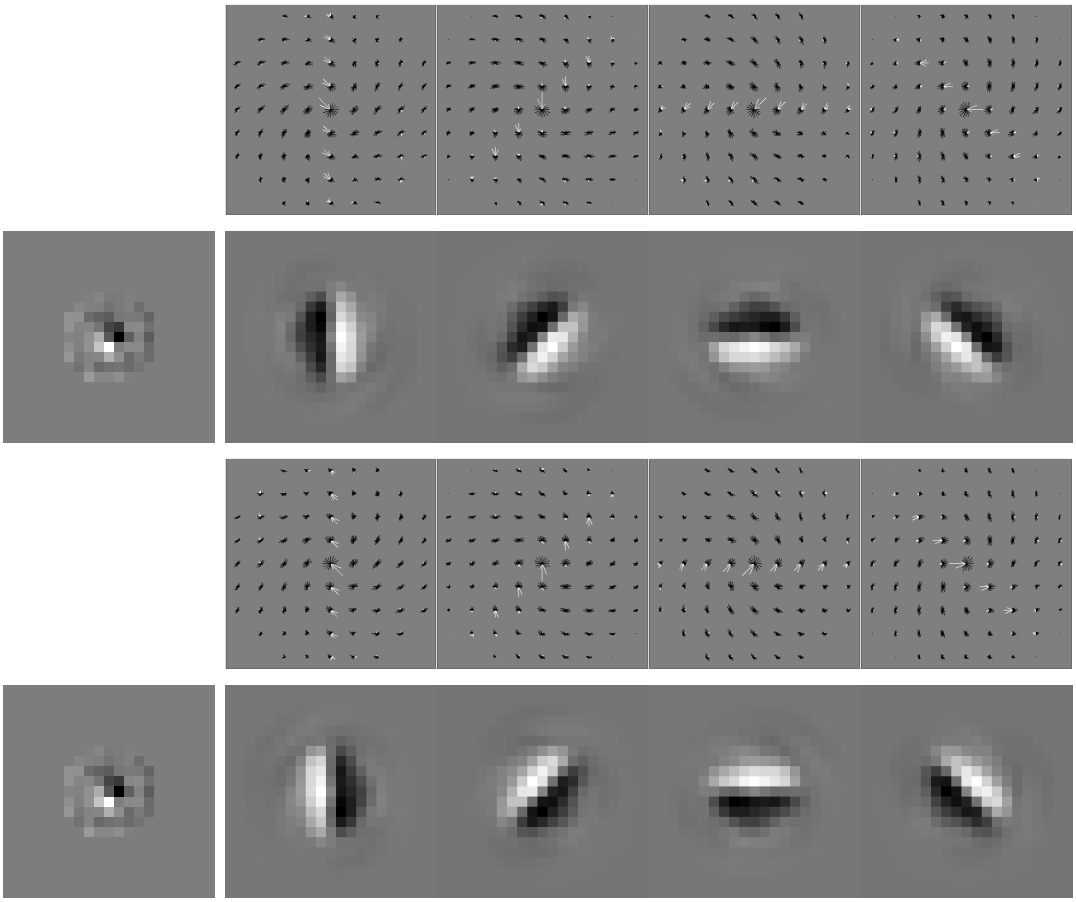


Figure 3.5: Rotational views of a second-layer weight kernel. 1^{st} and 3^{rd} row from the top: 8 rotational views (thinned from a set of 16) of a second layer weight kernel (displayed as in Fig. 3.4) . 2^{nd} and 4^{th} row from the top: The leftmost panel shows the first-layer basis feature; the other panels show the (linearly combined) first layer basis projections of the 8 second-layer features visualized above.

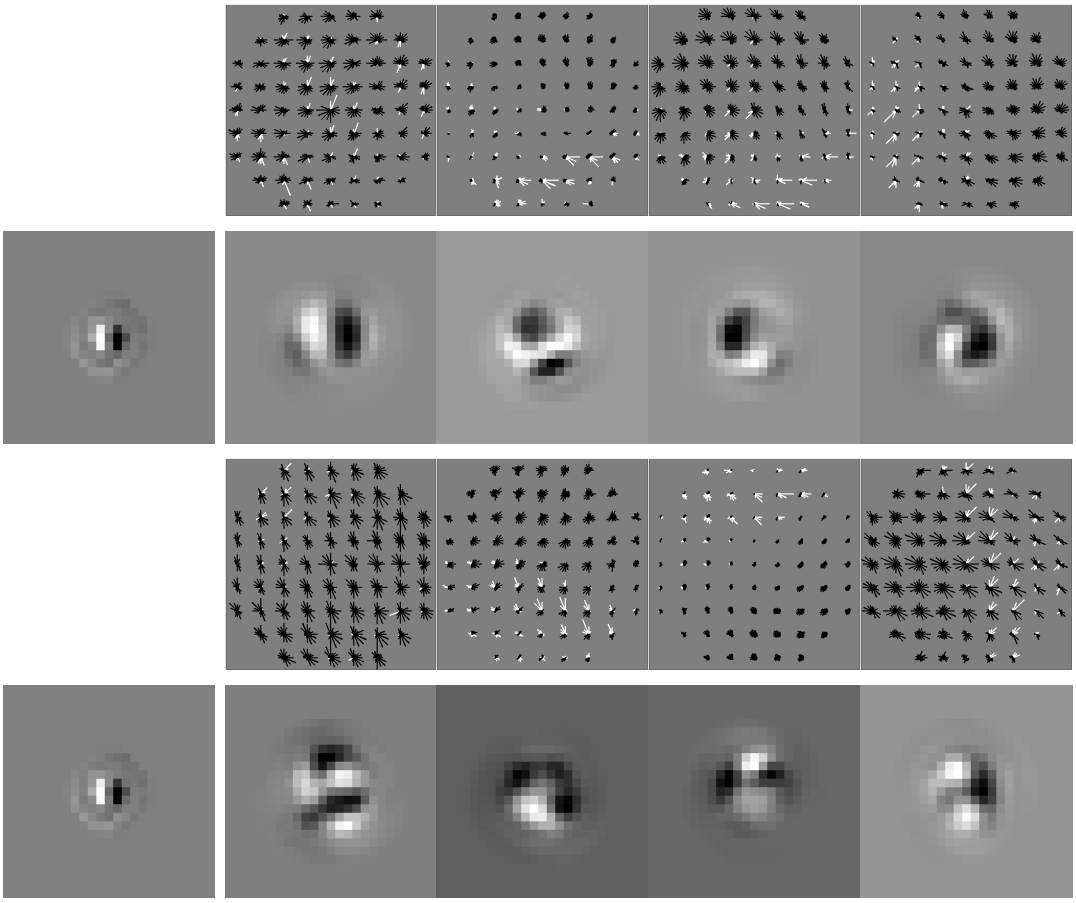


Figure 3.6: 1st and 3rd row from the top: The second layer weights (displayed as in Fig. 3.4) for 8 second-layer features. 2nd and 4th row from the top: The leftmost panel shows the first-layer basis feature; the other panels show the (linearly combined) first layer basis projections of the 8 features visualized above.

responses corresponding to an image patch can be augmented with latent variables to handle signal amplitude and dominant orientation. This allows e.g. modelling of oriented texture at arbitrary rotations. The learned edge filters at the first level of our model are analogous to the wavelet responses, while our second level units model the correlations between the coefficients. However, note (i) that the OAGSM model is only a model for patches not entire images, and (ii) that it does not provide a mixture model over the types of higher-level regularity, e.g. lines, corners, T-junctions etc. On the other hand the real-valued modelling of wavelet coefficients by OAGSM is more powerful than the binary activations of units in the STEER-DBN.

Our goal is to build in equivariance to known (translational and rotational) transformations. In contrast Memisevic and Hinton [2010] describe how to *learn* transformations based on pairs of training images using factored 3-way Boltzmann machines. Such a network could be used in to identify rotated versions of a given pattern, e.g. by fixing a reference version of the pattern and inferring the transformation. However, it seems rather excessive to learn the machinery for this when it can be built in. Our work should not be confused with the directional unit Boltzmann machine (DUBM) network of Zemel et al. [1995]. Although DUBM units contain a rotational variable, this is not used to model relative rotations of subcomponents. For example in Mozer et al. [1992] the authors present a convolutional architecture where the rotational variable denotes the phase of an oscillator, relating to the theory of binding-by-synchrony.

3.6 Discussion

As we have shown, the STEER-DBN architecture handles translation and rotation invariances. The other natural transformation to consider is image scaling. However, this can be relatively easily handled by the standard computer vision method of downsampling the input image by various factors, and applying the similar processing to each scale. Higher layers at a given scale can also take inputs from various scales. Alternatively one could introduce scaling assignment variables for each unit similar to the ones for rotation, scaling the features. These transformations would be *always local*, and choosing over several kinds of local transformations conforming to our framework has been later investigated in Sohn and Lee [2012]. Our work above and in Kivinen and Williams [2011] considers al-

ready local transformation with respect to in-plane rotation with the translation and rotation equivariant models.

Also later to our work Schmidt and Roth [2012] propose a related but different approach to encode transformations into an RBM. They demonstrate state-of-the-art results on rotation-invariant recognition and detection tasks, and suggest the usefulness of transformation equivariant models (as also ours) for categorization purposes. Possible future work not already mentioned includes learning more hidden layers, and using more expressive bottom-layer models, such as those allowing dependent Gaussian distributions for the visibles conditional on the hidden units [Ranzato et al., 2010b].

Chapter 4

Multiple Texture Boltzmann Machines

4.1 Introduction

The previous chapter focused on models constrained to have certain equivariance properties, and in the context of natural images such properties were built into Gaussian-Bernoulli restricted Boltzmann machines with convolutional feature sharing. In this chapter we focus in building effective generative models for visual textures based on more general Boltzmann machines.

One of the most flexible models for natural images proposed in the literature is the recent Product of Student-t Experts (PoT) with non-zero means (mPoT) [Ranzato et al., 2010b], which includes third-order interactions between visible and hidden units. Visual analysis of the samples drawn from the model (with tiled-convolutional weight sharing) as shown in Ranzato et al. [2010b, Figure 3] suggests that although piecewise smooth segments with clear intensity discontinuities at their borders can be generated, the model (as trained on natural image patches) does not hallucinate textured regions. The flexibility of the model can be increased by adding additional layers of hidden units, but the samples shown in Ranzato et al. [2011b, Figure 3] again do not show textured regions. These results suggest it is over-optimistic to expect a single mPoT model to be able to generate the wide variety of textures seen in natural scenes. However, such a model could be effective for a sufficiently small subproblem such as modelling individual textures. The generation of visual texture is a necessary sub-component of any credible model for visual scenes.

This chapter has three main contributions. First, we assess the power of the mPoT as a model for textures by specifically training on this task. A key advantage of the texture task is that one can assess the generative performance directly e.g. using the texture similarity score from Heess et al. [2009]. In contrast, current quantitative assessment of generic natural image models is typically based on discriminative performance of a classifier using features derived from the models, which is a very indirect way of evaluating generative performance. As well as assessing the texture modelling power of the mPoT with tiled-convolutional weight sharing, we also analyze the relative contributions of the mean and covariance parts of the mPoT by comparing its performance to those of its subcomponents, tiled-convolutional versions of the PoT/FoE and the Gaussian-Bernoulli restricted Boltzmann machine (GB-RBM). Our results suggest that while state-of-the-art or better performance can be achieved using the mPoT, similar performance can be achieved with the mean-only model.

Secondly, we develop a Boltzmann machine which is able to generate multiple textures; a natural extension of the model for specific textures. The model modulates a set of parameters shared across multiple textures with texture-specific parameters to create appropriate texture features. We compare the multi-texture model to single-texture models for constrained and unconstrained texture synthesis, and demonstrate comparable performance of the multiple texture model to individually trained texture models.

Thirdly, we will develop a method for generating globally varying textures, and applying texture interpolation, based on the multiple texture framework.

We begin in Section 4.2 by briefly reviewing the models for individual textures. We then describe the experimental setup for the analysis of these models, including data, assessment methods, modelling and inference details in Section 4.3. This section also gives the results, where we evaluate and analyze the performance and suitability of the methods as models of textures. Section 4.4 then develops a multi-texture Boltzmann machine and analyzes its properties, including a comparison of the generative performance of the model to those of single-texture models. Section 4.5 discusses texture morphing methods, and develops an approach for generating globally varying textures, and a texture interpolation method based on the multiple texture Boltzmann machine framework. Section 4.6 provides a summary and discussion.

4.2 Modelling of Individual Visual Textures with Boltzmann Machines

In our experiments we consider three models based on Boltzmann machines: the Gaussian-Bernoulli RBM, the Product of Student-t Experts, and the Product of Student-t Experts with non-zero means - all of which model the visible units $\mathbf{v}$ as normally distributed conditional on the hidden units $\mathbf{h}$; this is a setup popular in the modelling of continuous-valued data, such as natural images. We will use the same notation as in Section 2.1.2.1 of Chapter 2 where the details related to the models are given. Although other models providing such parameterizations do exist, these three models provide the typical spectrum of structure, namely whether the mean is constrained to be zero or not, and whether the covariance matrix is constrained to be diagonal or not. In order to scale up to large images, we use tiled-convolutional weight sharing (see Section 2.1.3 of Chapter 2), as in Ranzato et al. [2010b]. Below we denote the GB-RBM, the PoT, and the mPoT using tiled-convolutional weight sharing as Tm, TPoT, and TmPoT respectively.

Of course texture modelling has a long history and is not restricted to Boltzmann machine models. One simple texture model is a Gaussian random field; for example Heess et al. [2009] consider a simplified FoE with quadratic potentials, which they call the Gaussian FoE (GFoE). The FoE was extended to use bimodal potentials in Heess et al. [2009] to create the BiFoE model, and their results show that this generally improved performance over the GFoE and FoE models. See Section 2.1.6 for more details on these models. In earlier work Zhu et al. [1998] proposed a model based on fixed (rather than learned) filters, but with non-parameteric potentials. Finally we mention the nonparametric texture synthesis method in Efros and Leung [1999]. This grows a patch of texture from a seed, but does so without an explicit generative model; instead it pastes in new pixels based on the match to a reference sample of texture.

4.3 Dissecting Boltzmann Machine Texture Models

In this section we analyze the performance of the conditionally Gaussian Boltzmann machine models in texture modelling. Below we first discuss the data used for the experiments, and then in section 4.3.2 give details of how the models were trained. Results for unconstrained texture synthesis are given in section 4.3.3,

and for constrained synthesis (inpainting) in section 4.3.4.

4.3.1 Data

The data used in the experiments were Brodatz-texture images[1]. We applied similar rescaling as in Heess et al. [2009]: the 640x640 textures were rescaled to either 480x480 or to 320x320, preserving all major texture features. We then normalized each texture to have zero mean, and applied global scaling so that the standard deviation of each texture was 4. This scaling was used because the parameter σ in eq. 2.8 controlling the standard deviation of the Gaussian distributions in the GB-RBM density was fixed to unity in our code; it is equivalent to setting $\sigma = 1/4$ in the GB-RBM energy (and rescaling $\mathbf{a}$) if the texture were normalized to unit variance. The evaluation metrics used for quantitative analysis are insensitive to these normalization steps. Each image was divided into a top half used for training, and a bottom half for carrying out testing.

4.3.2 Learning

The training data consisted of patches of size 98×98 randomly cropped out of the the preprocessed training textures, and processed in batches of size 64. We experimented with several receptive field sizes, and the number of hidden units for the models. Increasing the number of hidden units typically improved the generative quality. We used a receptive field size of 11×11, and the tiling was done diagonally with a stride of one pixel. Thus there are 11 sets of filters (one for each offset), and we used 32 filters per set for both the mean $\mathbf{h}^m$ and covariance $\mathbf{h}^c$ hidden units, when applicable. We held the visible unit biases $\mathbf{a}$ of the Tm-models fixed to zero.

All of the models were trained by approximate maximum likelihood, using stochastic gradient ascent based on Fast Persistent chains Contrastive Divergence (FPCD) [Tieleman and Hinton, 2009] (see also Section 2.3.4.2). The implementation for training the models heavily used the code by Marc'Aurelio Ranzato[2], especially for the tiled-convolutional mPoT. We will now describe the main details of the learning algorithms: The hidden variables in the TmPoT energy can be integrated out analytically to give the free energy, as in Ranzato et al. [2010b,

[1] `http://www.ux.uis.no/~tranden/brodatz.html`.

[2] `http://www.cs.toronto.edu/~ranzato/publications/mPoT/mPoT.html`

eq. 6]. Parameter learning can be then done by computing the difference of positive and negative phase expectations of the free-energy gradients. In our learning procedure we initialized the negative particles, which are used in the computation of the negative phase, to zeros. They were updated during each iteration using a single-step of hybrid Monte Carlo (HMC) [Neal, 1996] (see also Sec. 2.3.3.5), which used a random momentum sampled from a zero-mean, spherical Gaussian, and applied 30 Leapfrog steps.

The models did not have any special boundary units, and therefore at the boundaries and especially at the corners (due to diagonal offsets between the tiles) there were sites which we less constrained than in the center of the image. This often caused boundary artifacts unless care was taken. We tried various ways of dealing with these problems, for each of the models and textures. The results we report use a mixed way of dealing with them: For all models except TPoT, we clamped the borders of the negative particles to zero as in Ranzato et al. [2010b]. For the TPoT, we simply discarded the boundary data in computing the gradients for parameter updates, which seemed to work best for this model. An analysis of the effect of different boundary handling methods to texture synthesis performance under the Tm-model is provided in Appendix C.1.2.

As in Ranzato et al. [2010b], the covariance filters of the TmPoT were pre-multiplied with a whitening transform matrix[3], and during training their L_2-norm was maintained at unity individually by normalization. The normalization avoids the decay of experts to zero, but removes scale adaption, the necessity of which is lessened by the whitening transform.

We initialized the mean weights $\mathbf{M}$ and covariance weights $\mathbf{C}$, and other parameters in general to small random values. The hidden biases $\mathbf{b}$ were, however, initialized to -2. We experimented with various parameter learning rates and combinations for the different models under different textures, but these did not matter much for reasonable ranges of values. For TPoT, we used equal learning rates of 0.001 for $\mathbf{C}$ and the parameters γ (see eq. 2.14). For Tm, we used a learning rate of 0.001 for $\mathbf{M}$, and 0.1 for $\mathbf{b}$. We used half of these learning rates for the parameters of the TmPoT models. The learning rates were held fixed for the fast parameters, but annealed for the regular parameters. We also used a small L_1-decay on the weights.

[3]We used texture-class specific ZCA whitening (see for example Bell and Sejnowski [1996]).

4.3.3 Unconstrained texture synthesis

Our quantitative analysis of generative performance first considered the quality of texture *samples*. To obtain samples from the models we ran HMC sampling for a large number of iterations, after which their states were stored for analysis. 128 samples of size 120×120 were collected for each model under each texture class. Texture patches and representative model samples (with boundary sites discarded) are shown in Figures 4.1 and 4.2. Visual inspection shows that while the samples from the TmPoT and Tm are good, TPoT clearly fails to provide a faithful model of the data. Sample filters learned for the different textures using the Tm are shown in Figure 4.6.

To provide a quantitative evaluation we compute the Texture Similarity Score (TSS) defined in Heess et al. [2009] between each sample and the testing texture patch. For a sample $\mathbf{s}$ and texture image $\mathbf{x}$ the TSS is defined as the maximum of normalized cross correlation (NCC) between them:

$$\mathrm{TSS}(\mathbf{s}, \mathbf{x}) = \max \left\{ \frac{{\mathbf{x}_{(1)}}^\top \mathbf{s}}{\|\mathbf{x}_{(1)}\|\|\mathbf{s}\|}, \ldots, \frac{{\mathbf{x}_{(\mathcal{I})}}^\top \mathbf{s}}{\|\mathbf{x}_{(\mathcal{I})}\|\|\mathbf{s}\|} \right\}, \tag{4.1}$$

where $\mathbf{x}_{(i)}$ denotes a patch within the image matching the size of $\mathbf{s}$, located at position i, and $\mathcal{I}$ denotes the number of possible patch locations. We used matching window of size 19×19 to compute the score, extracted from a random location in each sample, which is the same size as those of the samples used in Heess et al. [2009] to compute their scores. (Note that the results in Heess et al. [2009] use the whole texture for both training and texting; in contrast we have a training/test split, see sec. 4.3.1.)

As in Heess et al. [2009], the textures considered for quantitative analysis were D6, D21, D53, and D77 (see top rows of Figure 4.1). Figure 4.3 (left) and Table 4.1 (top) show a summary of quantitative analysis results based on the TSS. The scores for the TmPoT and the Tm are excellent for all the textures, even for the D77, which appears the least homogenous of the textures. While performance of the TmPoT for most textures the highest, it is closely matched by the Tm. The scores for the TPoT are much worse than those of either of the above models; this is consistent with the inability of the FoE model (convolutional PoT) considered in Heess et al. [2009] to produce high-quality texture samples. The fact that TPoT doesn't distinguish between $\mathbf{v}$ and $-\mathbf{v}$ is not the only reason for its poor performance in synthesis: scores obtained using absolute values of NCC are still significantly lower for TPoT than for other models (Abs-TSS sample

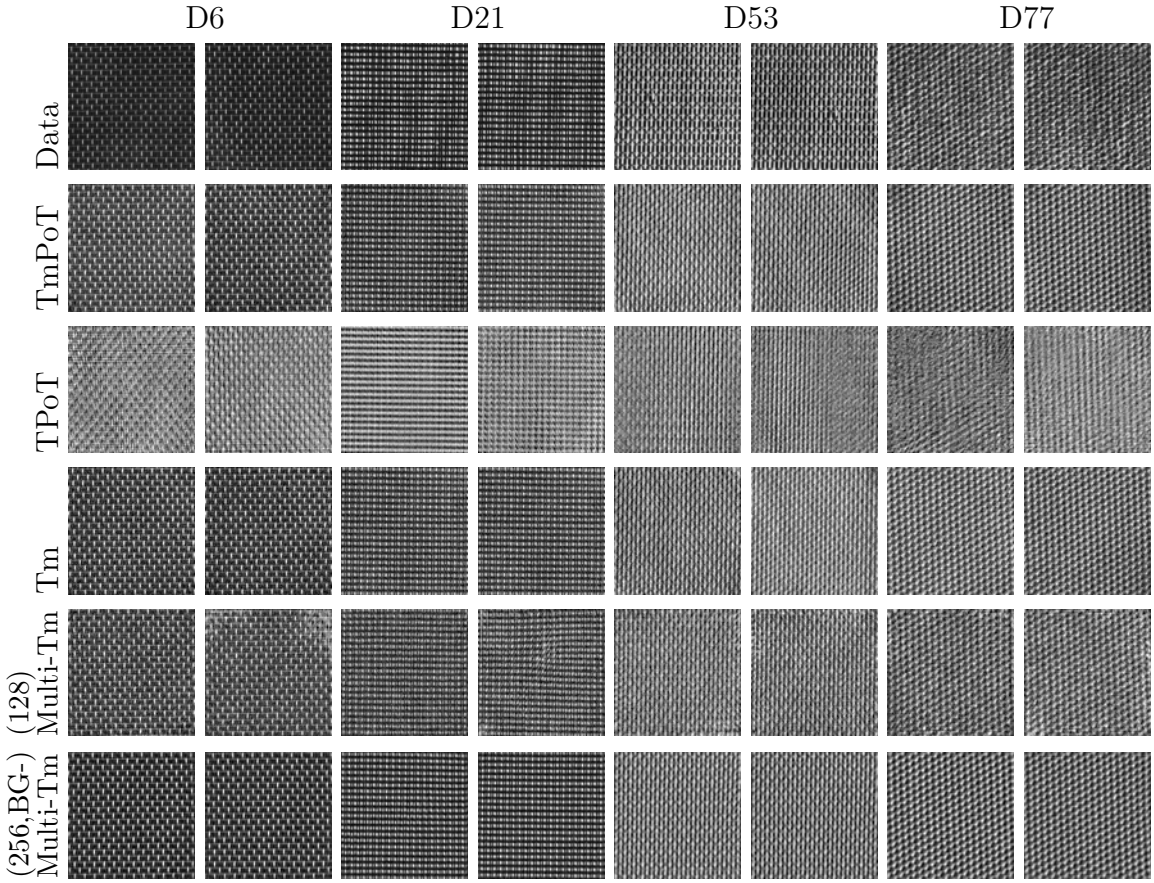


Figure 4.1: Example data patches (top row), and model samples (other rows), with each case scaled independently to cover the full intensity range.

mean±stds: D6: 0.6207 ± 0.0670, D21: 0.8214 ± 0.0591, D53: 0.8173 ± 0.0975, D77: 0.7080 ± 0.0796).

When comparing these results to Figure 3(a) in Heess et al. [2009], note that the filters used there were 7×7, and that 9 sets of filters were used, in a fully convolutional rather than (diagonally) tiled-convolutional fashion. Although many more parameters need to be learned for our models, within each 11×11 block in the image (except for boundaries) there were only 11×32 experts due to the diagonal stride between tiles used in our experiments; this is clearly less than $11 \times 11 \times 9$ experts of Heess et al. [2009]. There are also the training/test split differences noted above. With these caveats we note that all mean TSS scores are clearly superior even with the Tm models over BiFoE, and the difference is particularly noticable for D6 and D77.

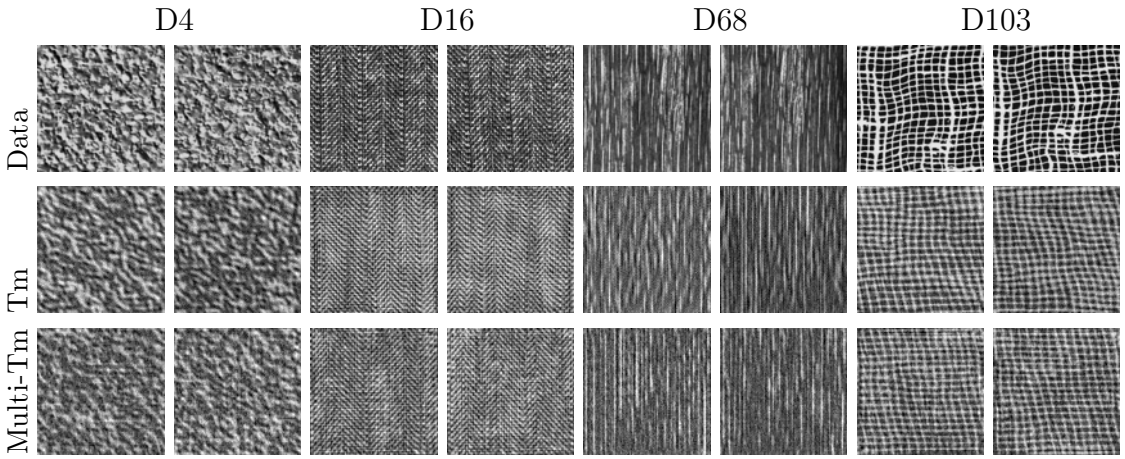


Figure 4.2: Synthesis results. Example data patches (top row) with representative results for Tm-models (middle row) and a multi-Tm (bottom row). Each case has been scaled independently to cover the full intensity range. The Multi-Tm model has 128 features per site.

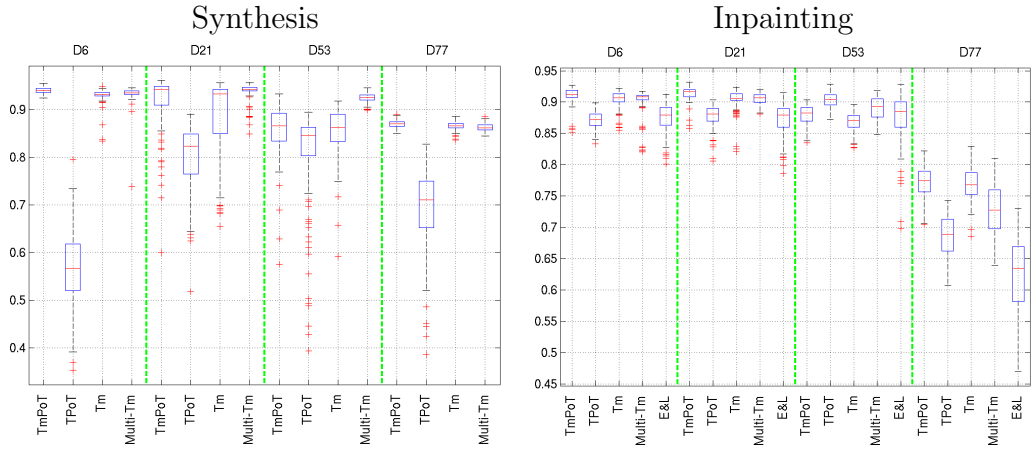


Figure 4.3: Quality assessment for the models, based on TSS/NCC between sample patches/inpainted area and corresponding Brodatz texture for unconstrained (left) and constrained (right) synthesis. The Multi-Tm model has 256 features per site. Boxes indicate the upper and lower quartiles as well as the median (red bar) of the TSS/NCC distributions; whiskers show extent of the rest of the data; red crosses denote outliers.

Synthesis (TSS):

	Brodatz Texture			
Model	D6	D21	D53	D77
TmPoT	0.9395 ± 0.0066	0.9182 ± 0.0554	0.8566 ± 0.0513	0.8693 ± 0.0072
TPoT	0.5651 ± 0.0792	0.8004 ± 0.0670	0.8051 ± 0.1026	0.6898 ± 0.0852
Tm	0.9299 ± 0.0149	0.8888 ± 0.0811	0.8548 ± 0.0475	0.8658 ± 0.0087
Multi-Tm (96)	0.8229 ± 0.1154	0.8897 ± 0.0507	0.8679 ± 0.0404	0.8191 ± 0.0287
Multi-Tm (128)	0.8886 ± 0.0932	0.9139 ± 0.0302	0.8914 ± 0.0353	0.8341 ± 0.0183
Multi-Tm (256)	0.9327 ± 0.0184	0.9377 ± 0.0172	0.9243 ± 0.0089	0.8623 ± 0.0082
Multi-Tm (256,BG)	0.9309 ± 0.0304	0.9383 ± 0.0155	0.9233 ± 0.0091	0.8624 ± 0.0082
Multi-Tm (256,BG-)	0.9502 ± 0.0323	0.9553 ± 0.0157	0.9413 ± 0.0091	0.8814 ± 0.0074
Tm (BG)	0.9312 ± 0.0125	0.8857 ± 0.0833	0.8430 ± 0.0689	0.8652 ± 0.0139
Tm (BG-)	0.9526 ± 0.0118	0.9020 ± 0.0848	0.8607 ± 0.0706	0.8869 ± 0.0140
Bi-FoE	0.7573 ± 0.0594	0.8710 ± 0.0317	0.8266 ± 0.0869	0.6464 ± 0.0215

Inpainting (NCC):

	Brodatz Texture			
Model	D6	D21	D53	D77
TmPoT	0.9106 ± 0.0144	0.9134 ± 0.0120	0.8785 ± 0.0162	0.7736 ± 0.0268
TPoT	0.8709 ± 0.0139	0.8770 ± 0.0182	0.9028 ± 0.0113	0.6867 ± 0.0303
Tm	0.9041 ± 0.0137	0.9041 ± 0.0175	0.8667 ± 0.0159	0.7698 ± 0.0280
Multi-Tm (96)	0.8793 ± 0.0217	0.8862 ± 0.0091	0.8514 ± 0.0203	0.6968 ± 0.0400
Multi-Tm (128)	0.8871 ± 0.0200	0.8931 ± 0.0092	0.8718 ± 0.0169	0.7112 ± 0.0453
Multi-Tm (256)	0.9010 ± 0.0214	0.9047 ± 0.0090	0.8901 ± 0.0178	0.7282 ± 0.0454
Multi-Tm (256,BG)	0.9017 ± 0.0208	0.9046 ± 0.0091	0.8898 ± 0.0178	0.7270 ± 0.0471
Multi-Tm (256,BG-)	0.9210 ± 0.0213	0.9224 ± 0.0090	0.9081 ± 0.0182	0.7451 ± 0.0487
Tm (BG)	0.9032 ± 0.0141	0.9040 ± 0.0179	0.8670 ± 0.0154	0.7707 ± 0.0260
Tm (BG-)	0.9240 ± 0.0140	0.9217 ± 0.0182	0.8860 ± 0.0156	0.7908 ± 0.0263
Efros&Leung	0.8746 ± 0.0239	0.8724 ± 0.0262	0.8732 ± 0.0412	0.6211 ± 0.0582
Bi-FoE	0.8769 ± 0.0163	0.8653 ± 0.0244	0.9145 ± 0.0125	0.6567 ± 0.0205

Table 4.1: Sample means and standard deviations of the texture synthesis (top) TSS- and inpainting (bottom) NCC-scores. We thank Nicolas Heess for providing the Bi-FoE results for the synthesis task. The inpainting results for Bi-FoE [Heess et al., 2009] are shown for rough comparison/indicative purposes, as they were obtained using a slightly different experimental setup. See Figure 4.7 and Table 4.2 for the inpainting results w.r.t MSSIM and TSS.

4.3.4 Constrained texture synthesis

We used an inpainting evaluation protocol very similar to Heess et al. [2009]: We took patches out of the test texture images, and created a square hole inside by setting texture values within the square to zeroes. The inpainting task was then to produce reasonable values to the zeroed out pixels. The quality was measured by the (i) NCC score with the ground truth region, (ii) mean structural similarity index (MSSIM) [Wang et al., 2004] (see also Appendix A.5) between the inpainted region and the ground truth region, and (iii) TSS between the inpainted region and the test portion of the Brodatz texture. Instead of using 70×70 images, we used 76×76 images and used a 54×54 instead of a 50×50 inpainting square as in Heess et al. [2009]. The reference frame border was then 11×11, compared to 10×10 of Heess et al. [2009]. The inpainting was done for the models by running HMC sampling, during which we constrained the reference border. The number of inpainting frames used in the experiments was 20 for each texture class, and the inpainting was done with 5 different random number generator initial states, producing 100 result images for each model under each texture class. We also compared against the nonparametric method by Efros & Leung (E&L) [Efros and Leung, 1999], and our implementation of that method used the training half of the image as the training data. The 'neighbourhood window' for infilling from the training data was 15×15, as used in Heess et al. [2009].

Example inpainting results for the models are shown in Figure 4.4. Inpainting results are summarized quantitatively w.r.t. NCC in Figure 4.3 (right) and Table 4.1 (bottom). Our experiments suggest that by providing a reference frame, the models are able to improve the quality of the samples as measured by the TSS[4] over those from texture synthesis[5]. As in the texture synthesis task, the scores for the TmPoT and the Tm model are highest in general, and comparable to each other. Interestingly, providing the reference frame provides a performance boost to the TPoT in relation to the other models: Although it still scores slightly lower than the other models on most textures, its performance is even slightly better than the other models for the D53 texture[6]. Comparing to the Efros & Leung and BiFoE results (last row of Table 4.1), we observe very similar results

[4]MSSIM cannot be used for assessing both of the tasks.

[5]The sampling and inpainting scores are not directly comparable because the patch sizes for scoring were different, and typically the smaller the patch the larger the score.

[6]The slightly worse performance of the TmPoT is likely to be related to local optima and boundary issues (which for that model were most problematic).

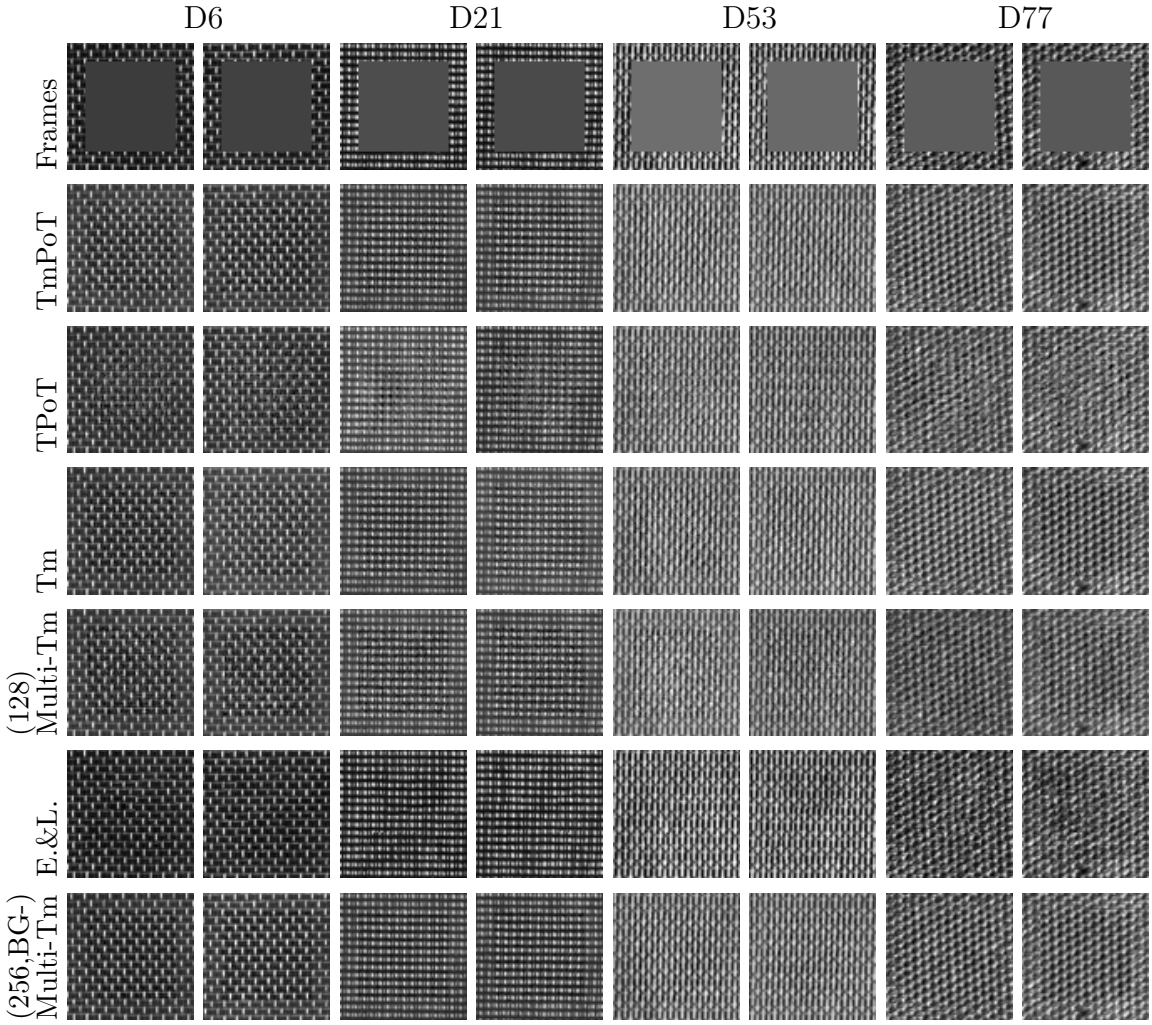


Figure 4.4: Inpainting results. Example inpainting frames (top row) with representative results for the models (other rows). Each case has been scaled independently to cover the full intensity range.

for D6, D21 and D53, but that our performance is markedly better for D77.

We have also analyzed the inpainting results w.r.t. MSSIM (computed between the inpainted region and the ground truth data for that region) and TSS (computed between the inpainted region and the test portion of the Brodatz texture) which are summarized in Figure 4.7 and Table 4.2. MSSIM is typically considered to be perceptually more valid than metrics based on mean squared error for assessing image quality in the task, and is widely used in assessing inpainting quality. The results have a similar pattern to the NCC results with respect to all textures except for D77, in the case of which the TPoT and the Efros&Leung scores are clearly lower/worse.

The following section develops a novel framework for Boltzmann machines to generate multiple textures, where we will use Tm as the base-model, since it obtained state-of-the-art results, and it is computationally much less complicated than the TmPoT[7].

4.4 Multi-Texture Boltzmann Machines

The above Boltzmann machine models are for individual textures. Here we describe a framework for multiple textures. Our model has two sets of parameters Θ; θ_{global} are shared parameters across the different classes, while the parameters $\{\theta_m\}$ are specific to each individual texture class $m = 1, \ldots, M$.

Let $\mathbf{V}_m$ denote the visibles of the images in texture class m. We assume that each of the M probabilities $p(\mathbf{V}_m \mid \theta_m, \theta_{\text{global}})$ are defined by Gaussian-Bernoulli RBMs. The weights $\mathbf{M}$ of these models are set to be global, while the biases $\mathbf{b}$ are set to be texture-specific, so that $p(\mathbf{V}_1, \ldots, \mathbf{V}_M \mid \Theta) = \prod_{m=1}^{M} p(\mathbf{V}_m \mid \mathbf{b}_m, \mathbf{M})$. Switching between the different classes is achieved by having a high-level categorical variable y with M states denoting the different textures. The appropriate biases are switched in by selecting the state of y, similar to the implicit mixture construction in Nair and Hinton [2009].

As in the previous experiments, we use tiled-convolutional weight sharing with the model in the following experiments. We denote this model the multi-Tm. To further motivate the multi-Tm, imagine that we start with a single-Tm model for a specific texture, and then add the filters from all the other texture models to the energy, but setting the biases for the filters from all of the models to large negative values. This will have the effect of "turning off" the filters from the other models, leaving in effect the original single-Tm model. Thus this model can be made to mimic each of the original single-Tm models by adjustment of the biases. However, the real multi-Tm can be more powerful by sharing filters across textures.

4.4.1 Learning

We learned the parameters of the models by approximate maximum likelihood, using stochastic gradient ascent based on Fast Persistent chains Contrastive Di-

[7]Boundaries were typically also easier to deal with it.

vergence (FPCD), as above. We assigned sets of 64 negative particles to each of the texture classes, which were updated with HMC sampling, so that for texture class m samples at an epoch were drawn from the model specified by current parameters $\mathbf{b}_m$ and weights $\mathbf{M}$, using similar techniques as described before. Our implementation loops over texture classes updating their negative particles which are then used to compute the gradients and parameter updates for their class-specific biases, and accumulates gradients w.r.t weights which are used to update the weights once all classes have been sweeped/statistics collected.

4.4.2 Experiments

We have trained models for 8 Brodatz texture categories (D6, D21, D53, D77, D4, D16, D68, and D103) with 96, 128, and 256 features for each tile set, shared over all of the textures; the models for individual textures (single-Tms) each have 32 specific features, totalling $32 \times 8 = 256$ features.

To investigate the specificity/generality of features of a multiple texture model we considered a 256-feature model trained on the full textures, and evaluated hidden unit activation probabilities of each feature with each of the bias settings (one per texture class) as a response to samples from each of the texture classes.

We then applied multi-class Fisher's linear discriminant analysis (see e.g. Bishop [2006, §4.1.6]) to these vectors to rank the features according to their separability/texture specificity, using the $J(\mathbf{W})$ criterion from Bishop [2006]. Thus for each feature type we aim to find linear projections $\mathbf{y} = \mathbf{W}\mathbf{x}$ of the feature-specific vectors $\mathbf{x}$ which would separate the texture classes as well as possible. The separation criterion is set to score high when the between-class covariance of the projections is high and within-class covariance is small. The particular form we use defines:

$$J(\mathbf{W}) = \mathbf{Tr}\left\{\left(\mathbf{W}\mathbf{S}_W\mathbf{W}^\top\right)^{-1}\left(\mathbf{W}\mathbf{S}_B\mathbf{W}^\top\right)\right\}, \tag{4.2}$$

where $\mathbf{Tr}$ denotes a matrix-trace operator, $\mathbf{S}_W$ denotes a within-class covariance matrix, and $\mathbf{S}_B$ denotes a between-class covariance matrix. The covariance matrices are defined as follows:

$$\mathbf{S}_W = \sum_k \sum_{n \in \mathcal{C}_k} \left(\mathbf{x}^{(n)} - \mu_k\right)\left(\mathbf{x}^{(n)} - \mu_k\right)^\top \tag{4.3}$$

$$\mathbf{S}_B = \sum_k N_k \left(\mu_k - \mu\right)\left(\mu_k - \mu\right)^\top, \tag{4.4}$$

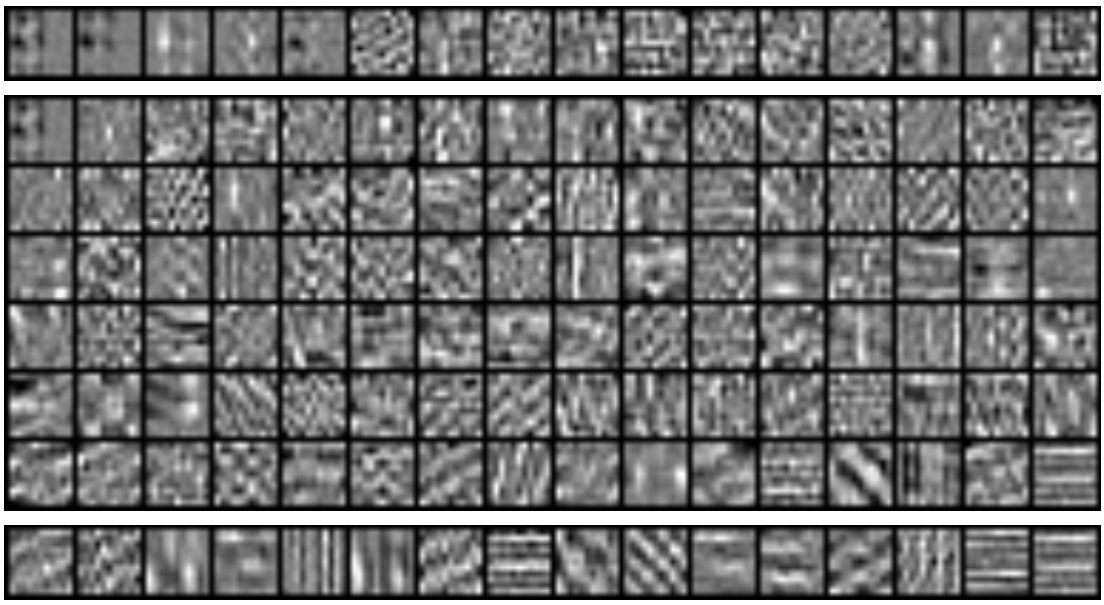


Figure 4.5: A sampling of weights of a multi-Tm, ordered from left-to-right and top-to-bottom with increasing Fisher LDA score in each block. The top block shows the features with the 16 smallest scores, the bottom block the 16 largest scores, and the middle block a thinned set of 128 features.

where N_k denotes the number of cases in class $\mathcal{C}_k$, $\mu_k = \frac{1}{N_k} \sum_{n \in \mathcal{C}_k} \mathbf{x}^{(n)}$ (class-specific average under a feature), and $\mu = \frac{1}{\sum_k N_k} \sum_n \mathbf{x}^{(n)}$ (average of all vectors under a feature). The optimization amounts to solving an eigenvalue problem; we solve $\mathbf{W}$ for each feature by computing the number of texture classes minus one eigenvectors of $\mathbf{S}_W^{-1}\mathbf{S}_B$ associated with the largest eigenvectors.

Based on this ranking, we visualize these features in Figure 4.5 so that the top row illustrates 16 least separable features, the block below it shows a thinned set of 128 features, and the row below it 16 most separable features, with increasing separability from left-to-right and top-to-bottom. Many of the most separable ones resemble filters in texture-specific (single-Tm) models for the same data as shown in Figure 4.6.

We have evaluated the sampling and inpainting performance of the models using the setup of previous section. Representative samples from the model with 128 features per site are shown in Figure 4.1 (bottom row), Figure 4.2 (bottom row) and Figure 4.4 (second row from the bottom). Visual inspection of the figures shows that they are comparable to those of the individually trained Tm-models, and that the models can capture the statistics of a wide variety of textures

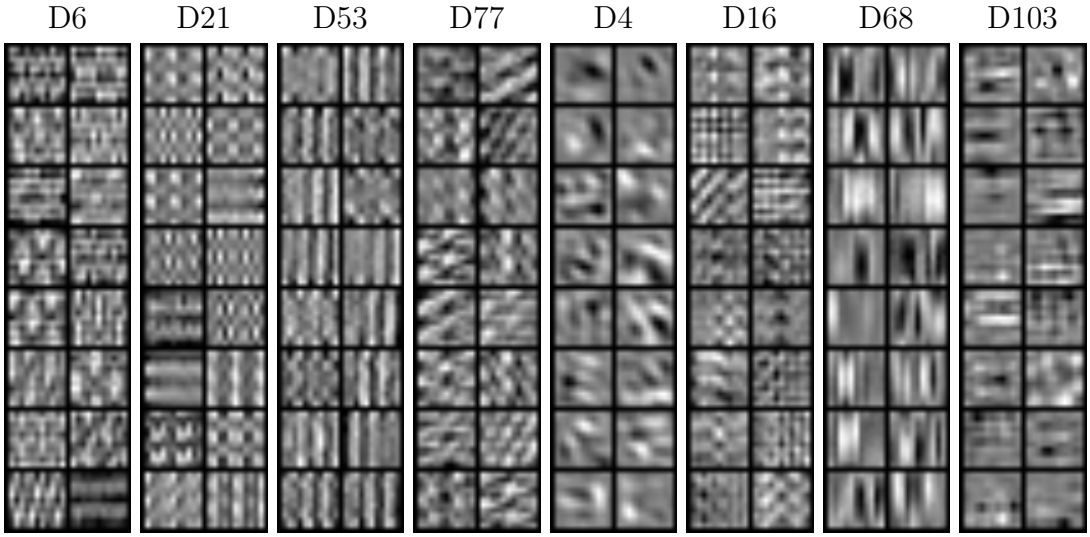


Figure 4.6: A sampling of weights of single-Tm-models.

effectively. Numerically the performances are also similar, as can be seen from Figure 4.3 and Table 4.1, with the exception of higher synthesis performance for multi-Tm models on D53, and higher NCC inpainting performance for the individually trained D77. Table 4.1 shows that the performance of the multi-Tm in general improves as the number of features is increased. The results w.r.t. MSSIM and TSS summarized in Figure 4.7 and Table 4.2 have a similar pattern to the NCC results with respect to all textures except for D77, for which the multi-Tm scores are now similar to those of the texture-specific TmPoT- and Tm-models, and (as mentioned before) the TPoT and the Efros&Leung scores are clearly lower.

Finally we consider two additional inference approaches for synthesis and inpainting with the Tm and the Multi-Tm (256) based on block-Gibbs sampling, denoted BG and BG-. BG sets the result texture images as the actual samples (in synthesis at iteration number 10,000 starting from a zero-initialization and in inpaiting at iteration number 5,000) whereas BG- sets them as the conditional means of visible units given hidden units (and only lacks the i.i.d. Gaussian (which is here standard) noise added to obtain the BG). The obtained results are summarized numerically in Tables 4.1 and 4.2. We can see from the table that (i) synthesis results with the baseline HMC and the BG are similar, and (ii) the BG- obtains clearly better results than the BG (and the baseline HMC). Appendix C.1.1 finds similar results on an experiment which also considers the

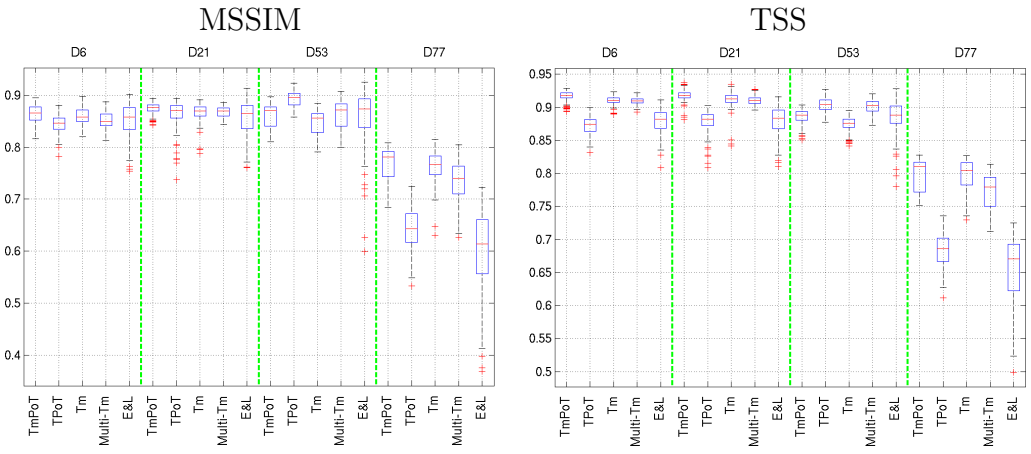


Figure 4.7: Constrained synthesis quality assessment for the models, based on MSSIM between inpainted area and corresponding Brodatz texture area (left) and TSS between inpainted area and the test portion of the Brozatz texture (right). The Multi-Tm model has 256 features per site. Boxes indicate the upper and lower quartiles as well as the median (red bar) of the MSSIM/TSS distributions; whiskers show extent of the rest of the data; red crosses denote outliers.

effect of the sampling method in learning single-texture models with a Tm. As also stated in the appendix, using the conditional mean as the estimator can be seen as applying *Rao-Blackwellization* [Blackwell, 1947]. These results have not been published in Kivinen and Williams [2012] and are novel.

4.5 Morphing and Interpolating Textures

We have demonstrated above with the multiple texture framework the synthesis of homogeneously textured images of several classes. We have also experimented with varying the biases in a spatial fashion, so to synthesize globally varying textures using a texture-by-numbers approach. In Figure 4.8 bias settings corresponding to two different textures have been used for manually specified image regions, including letters and the background; the model transitions nicely between the two textures[8]. Motivated by the effective performance, we now extend the methods further to consider even more complicated texture analysis problems, called texture morphing. Texture morphing includes several areas including

[8]Samples were drawn by HMC.

Inpainting (MSSIM):

	Brodatz Texture			
Model	D6	D21	D53	D77
TmPoT	0.8629 ± 0.0175	0.8748 ± 0.0108	0.8607 ± 0.0228	0.7668 ± 0.0322
TPoT	0.8447 ± 0.0176	0.8624 ± 0.0284	0.8932 ± 0.0143	0.6419 ± 0.0396
Tm	0.8591 ± 0.0167	0.8666 ± 0.0179	0.8482 ± 0.0234	0.7632 ± 0.0316
Multi-Tm (96)	0.8278 ± 0.0172	0.8509 ± 0.0101	0.8292 ± 0.0301	0.7030 ± 0.0376
Multi-Tm (128)	0.8346 ± 0.0152	0.8568 ± 0.0097	0.8471 ± 0.0249	0.7194 ± 0.0391
Multi-Tm (256)	0.8508 ± 0.0161	0.8682 ± 0.0096	0.8642 ± 0.0251	0.7345 ± 0.0399
Multi-Tm (256,BG)	0.8511 ± 0.0165	0.8682 ± 0.0093	0.8635 ± 0.0251	0.7340 ± 0.0427
Multi-Tm (256,BG-)	0.8770 ± 0.0170	0.8901 ± 0.0090	0.8851 ± 0.0252	0.7555 ± 0.0443
Tm (BG)	0.8586 ± 0.0164	0.8663 ± 0.0184	0.8488 ± 0.0224	0.7645 ± 0.0282
Tm (BG-)	0.8853 ± 0.0163	0.8878 ± 0.0188	0.8705 ± 0.0225	0.7861 ± 0.0290
Efros&Leung	0.8524 ± 0.0318	0.8566 ± 0.0344	0.8558 ± 0.0578	0.6012 ± 0.0760

Inpainting (TSS):

	Brodatz Texture			
Model	D6	D21	D53	D77
TmPoT	0.9173 ± 0.0073	0.9179 ± 0.0087	0.8859 ± 0.0117	0.7992 ± 0.0240
TPoT	0.8722 ± 0.0135	0.8782 ± 0.0178	0.9029 ± 0.0108	0.6846 ± 0.0243
Tm	0.9103 ± 0.0066	0.9114 ± 0.0138	0.8740 ± 0.0123	0.7983 ± 0.0210
Multi-Tm (96)	0.8873 ± 0.0085	0.8939 ± 0.0067	0.8610 ± 0.0137	0.7484 ± 0.0272
Multi-Tm (128)	0.8956 ± 0.0069	0.8998 ± 0.0059	0.8800 ± 0.0101	0.7597 ± 0.0187
Multi-Tm (256)	0.9096 ± 0.0054	0.9106 ± 0.0062	0.9010 ± 0.0116	0.7720 ± 0.0276
Multi-Tm (256,BG)	0.9099 ± 0.0058	0.9106 ± 0.0063	0.9009 ± 0.0115	0.7726 ± 0.0301
Multi-Tm (256,BG-)	0.9298 ± 0.0056	0.9286 ± 0.0060	0.9192 ± 0.0119	0.7920 ± 0.0313
Tm (BG)	0.9098 ± 0.0069	0.9114 ± 0.0140	0.8740 ± 0.0124	0.7989 ± 0.0195
Tm (BG-)	0.9307 ± 0.0068	0.9289 ± 0.0144	0.8930 ± 0.0126	0.8198 ± 0.0196
Efros&Leung	0.8789 ± 0.0194	0.8789 ± 0.0219	0.8843 ± 0.0261	0.6541 ± 0.0533

Table 4.2: Sample means and standard deviations of the texture inpainting MSSIM-(top) and TSS-scores (bottom).

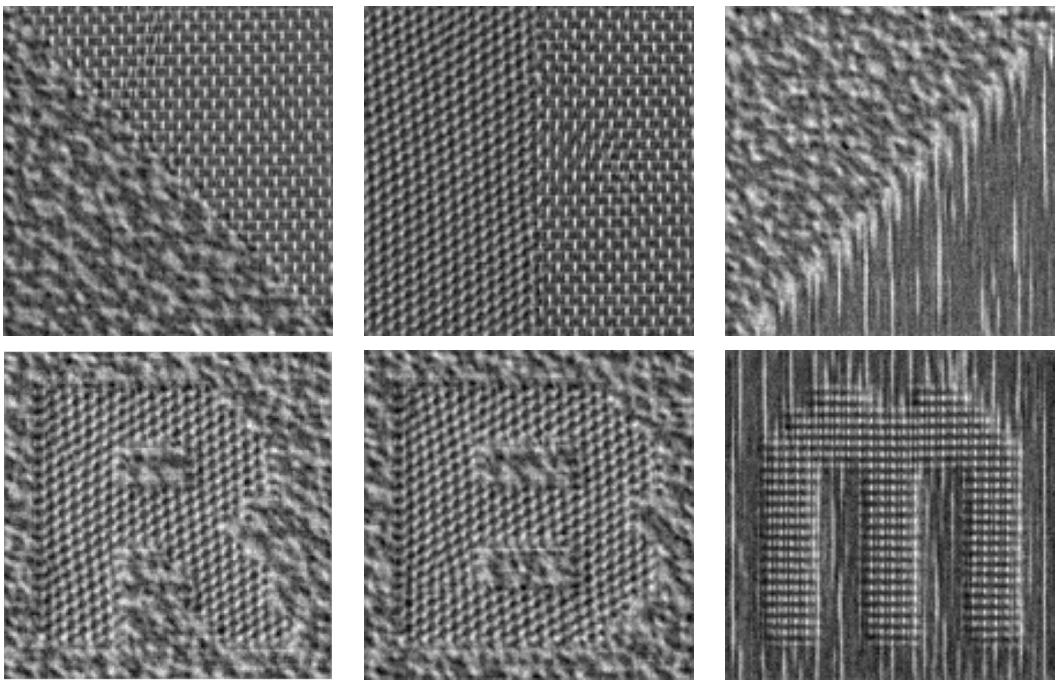


Figure 4.8: Texture results obtained by using different bias settings for different manually specified regions, including letters and the background.

texture blending, texture metamorphosis (Liu et. al. 2002), and texture interpolation (Ruiters et. al. 2010). In texture blending the goal is to create a novel texture based on two or more source textures, so that the resulting texture has features from the different source textures; the generated texture can be seen as a blend of the source textures. In texture metamorphosis, the task is to generate a sequence of texture images, from a source texture to a target texture, with smooth transitioning between the texture images. Texture interpolation takes the problem of texture metamorphosis to the generation of spatially varying texture, with the task of filling in missing data in an image with data from multiple textures.

So far we have not explored the synthesis of novel texturing possibly needed in all of the tasks, or creating smooth texture transitioning in the synthesis for texture metamorphosis and texture interpolation. In the following we will extend our earlier methods for enabling also these remaining components, and demonstrate their effectiveness in several example problems.

4.5.1 Spatially interpolated bias fields

The basic idea is to create spatially interpolated bias fields by forming convex combinations of the biases at each location. The energy-function of the model used to generate the texture images and scenes is defined as follows:

$$E(\mathbf{h}, \mathbf{v}) = \sum_i \frac{(v_i - a)^2}{2\sigma^2} - \sum_{s,t} \sum_{j=1}^{J} h_j^{s,t} \left(\sum_{k=1}^{K} \pi_k^{s,t} b_k^{s,j} + \frac{1}{\sigma^2} \mathbf{M}^s(\cdot, j)^\top \mathbf{v}_{\mathcal{N}_{s,t}} \right), \quad (4.5)$$

where π denotes so-called hidden unit bias combination weights, with K different values for each filter application position , indexed by diagonal offset s for the different tilings and patch index t (which together identify the patch location within the visibles lattice). The other parameters are those of a tiled-convolutional multiple-texture Gaussian RBM, with $\mathbf{M}$ denoting the weights of a multiple texture model, and $\{\mathbf{b}_k\}_{k=1}^K$ the K sets of training texture-class specific biases associated with them. In this example, the visible bias a, and σ is common to all of the texture classes.

We can view the models used in earlier experiments using categorical hidden unit bias combination vectors, and the category was either fixed (to generate homogeneously textured images) or varied spatially (to generate globally varying texture images). In the following experiments we will be considering tasks where the combination vectors are not categorical, but will be from a $K-1$ simplex[9], with the same or different values for the different positions, depending on the application.

In blending textures k and m the combination vectors are defined as follows: $\pi_\ell^{s,t} = 0.5\delta(\ell \in (k, m), 1) \ \forall \ s, t$, and so the *effective* hidden unit biases of the model will be the average of those associated with training texture k, and m. The rationale is that the biases affect the activation probabilities of features common to all textures, and for a blend the features should be active with probability according to the average of those associated with the source pairs classes, and their biases should be thus averaged. In texture interpolation, the combination weights are based on interpolation coefficients, varying spatially from the ground-truth categorical combination vectors defined by the existing texture data. We used bi-linear interpolation in our experiments.

[9]Meaning the bias combination vectors per site define convex combinations.

4.5.2 Related work

We discuss here closely related work to ours, see Ruiters et al. [2011] for a review of texture interpolation methods, and Wei et al. [2009] for a recent review of state-of-the-art methods for example-based texture synthesis for related other methods. Many of the texture morphing and interpolation methods are based on using image warping methods, and in general image metamorphosis/morphing methods. One of the earliest of these approaches is that of Liu et al. [2002], who introduced the problem of texture metamorphosis/morphing. Similar to typical image morphing methods, the approach divides the problem of obtaining a smooth morphing sequence of frames into two problems (i) of making correspondences between the source and the target images, and (ii) the problem of estimating a warping function to accommodate the changes in a smooth manner. Due to the complexity of making correspondences for even typical texture scenes, manual user assistance is used by specifying patterns (patches) in the source and in the target images, and specifying correspondences of features in them using landmarks. Based on this information, similar patterns are detected in the full images, for which correspondences are made automatically. The correspondence is formulated as an integer programming problem, and solved using the Hungarian algorithm. The warp function is then obtained by combining the pattern and the landmark correspondences, and is estimated by using a sparse points interpolation technique – a standard morphing path estimation algorithm.

Matusik et al. [2005] also consider a warp-based method for morphing textures, but in contrast to the approach in Liu et al. [2002], the correspondence estimation for morphing is fully automatic. Key to this is the analysis of morphability/similarity between textures, as measured by the residual error of a warping method optimizing the feature alignment between the textures. Based on this metric, a simplicial complex is created, with nodes of the graph representing different textures, and connections between nodes denoting that a morphing can happen between the associated textures, which is the case if the respective warping residual error is small enough. After the initial model is built, novel textures (which become novel nodes) can be then morphed from the nodes of the graph within a clique with a warping method, by linearly blending all pairwise morphs of the clique node textures, with convexly combined warps between each source node and its target nodes. The warp computation is done on feature map

representations of the textures, in a coarse-to-fine search in which smoothness is encouraged using a regularization technique. The method also includes sharpness preservation by a histogram matching method, operating on steerable pyramid coefficients. High-frequency statistics preservation via similar techniques is used also in other works, including in Ruiters et al. [2011]. In the paper a simplicial complex of roughly 1500 nodes is built (from roughly 1500 textures of size 128 x 128, rotated and scaled to have approximately the same orientation and feature size) using approximations for computing the graph connectivity.

The local homogeneity of statistics under globally varying textures has motivated several MRF-based or -inspired synthesis approaches. Many of these are texture resampling based, inspired by the nonparametric texture resampling method in Efros and Leung [1999] (or extensions such as that in Efros and Freeman [2001]), with local search using some similarity metric to feed the texture synthesis for the positions. The image analogies work [Hertzmann et al., 2001] considers several modalities of image data to create analogies with respect to the modalities, with the search performed over the joint modality representations. The approach has been adopted in several methods for globally varying texture synthesis, including in Liu et al. [2004], and in Zhang et al. [2003]. Zhang et al. [2003] consider several control maps: a texton mask defining the textons, an orientation field defining the orientation of progression, and transition function defining the rate of texture change/progression, each of which are defined manually.

Ruiters et al. [2011] combine several techniques proposed in the field in their method, mostly extending the approach of Kwatra et al. [2005]. To interpolate two textures, their method performs nearest-neighbor search for corresponding neighborhoods from manually defined binary feature maps for the textures, from both of the sets. The approach then interpolates the neighborhoods, which includes color adjustment (so-called α-blending) and feature warping, and synthesizes them using a multiscale approach of Kwatra et al. [2005], with the addition of several ad-hoc techniques to obtain effective visual quality, as is common with many of the approaches in the field.

The image melding method [Darabi et al., 2012] is a state-of-the-art texture interpolation method, which is also one of the most unified and self-contained ones. For a target interpolation position, similar neighborhoods are searched for using not only from the raw textures, but also from their geometrically and

photometrically transformed sets, and in the joint representation of them and their (automatically extracted) gradient information. In texture interpolation, multi-modal neighborhoods are searched from the source sets, similar to Ruiters et al. [2011], here by optimizing an energy function of a convex combination of source-specific energies, each consisting of weighted distances in terms of the colour and gradient data. The target image is then set according to a screened Poisson equation solver.

Similar to the method of Darabi et al. [2012], our approach is fully automatic, does not need user interaction, does not use warping, and is created with a global consistency criterion. Our approach can use data/features from images outside of the textures being interpolated, whereas the image melding approach in Darabi et al. [2012] relies only on the two source textures, although it could in principle be adapted to such settings easily. In contrast to several other methods, the model features in our case are learned from the data, rather than hand-engineered. Similar to most state-of-the-art methods, the interpolation is not done on the actual pixel values. In many of such methods it is done at the feature level, but in our case on the probability distributions over the feature states/activations. Furthermore our model also defines a fully specified generative model to accomplish the texture generation.

4.5.3 Texture interpolation experiments

By using convex combinations of the biases corresponding to pairs/several different textures suggest that plausible novel textures to those considered in training can be generated from the multiple texture model. A few such cases are shown in Figure 4.9, where each middle column image has been synthesized from a model with biases set as the average of two training texture specific biases used in the synthesis of the images on the left and on the right of the case[10]. All of the images produced by the model with averaged biases differ from those produced by a model with any of the training texture specific biases.

We have also experimented synthesis by spatially varying the biases using the spatial interpolation mechanism. Our results suggest that both abrupt texture changes and smooth morphs are possible based on appropriate spatial adjustments of the multiple texture model biases. These can be also seen from the

[10]The simulation was according to the BG- approach, as with the rest of the results in this chapter.

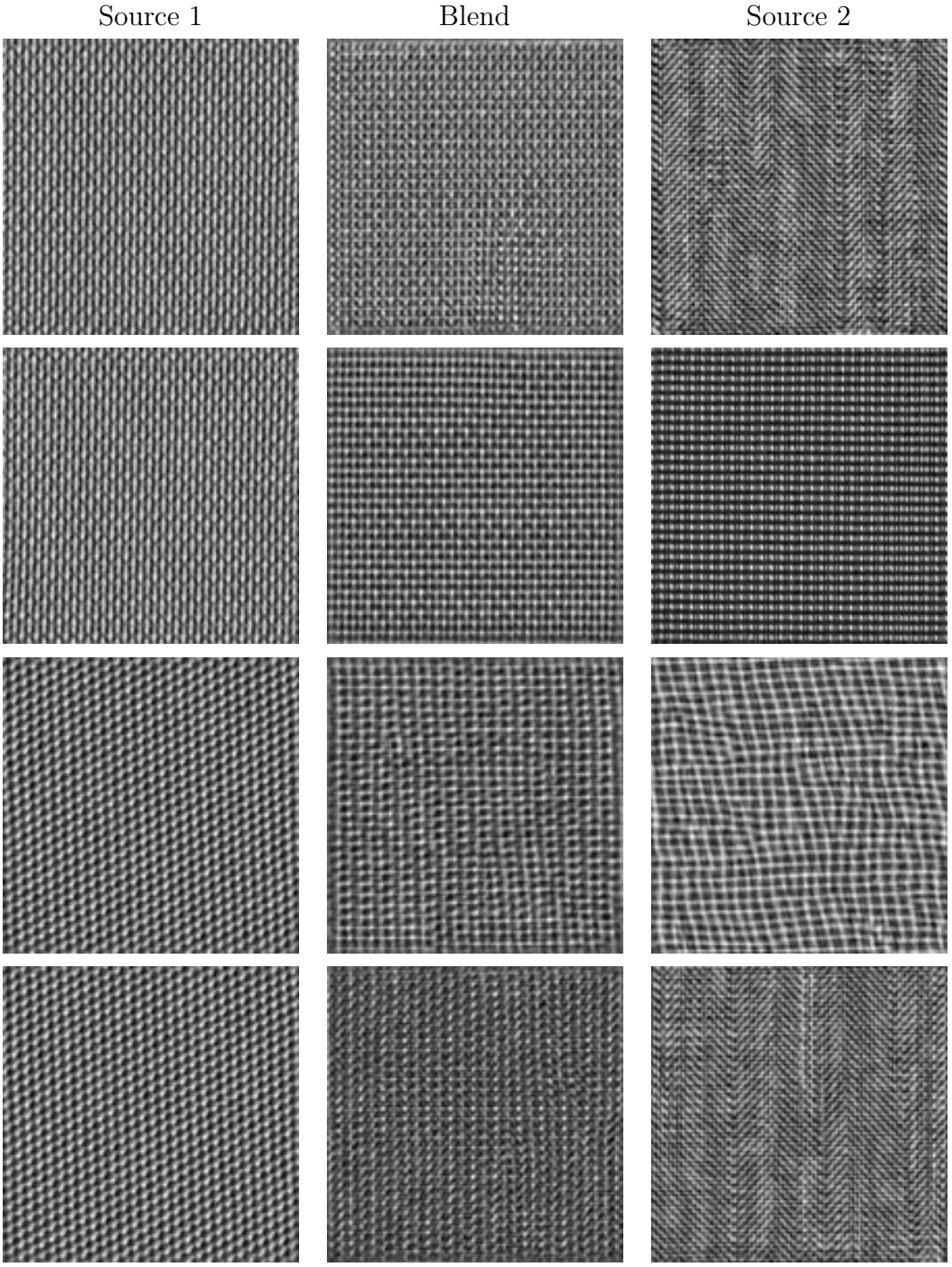


Figure 4.9: Example texture synthesis results obtained by a tiled convolutional Gaussian Bernoulli RBM (Tm) with weights of the multi-Tm of [1] with 256 features per site, trained on eight Brodatz textures, and biases set either to texture-specific biases, or set as the average of two texture-specific biases. The leftmost and rightmost columns show samples by using texture-specific biases. The middle column shows samples obtained using the average of the texture-specific biases used for the images on the left and on the right of each case.

Figures 4.10, 4.11, and 4.12. In Figure 4.10 the biases used in the synthesis of the middle column images are interpolated horizontally between two training texture specific biases which have been used in the synthesis of the images on the left and right. The middle column images demonstrate smooth texture morphing, and in some cases the creation of a novel texture. In Figures 4.11 and 4.12 the bias field used in the synthesis of the middle column images are again based on spatial interpolations between two training-texture-specific biases. Images synthesized with the training-texture-specific biases are shown on the left and on the right of each morphing case. The mixing proportions of the texture specific biases used in the morphing are visualized at the top row. In Figure 4.11 the bias field specification was such that the two sets of biases were used at the corners so that the biases of each pair of neighboring corners were different from each other (and so the biases were the same for the opposing corners). The biases at the other locations were interpolated from those at these corners by using bilinear interpolation. In Figure 4.12 the mixing weights were specificied in a more complicated way, but also using bilinear interpolation from control point values. Notice here how both clear texture borders and smoothly varying texture changes are generated.

We also considered source and target textures being of the same texture type, but associated with different viewpoints and/or illumination conditions, focusing on a texture from the CUReT database. Figure 4.13 shows raw texture data, and synthesized samples from a multi-Tm learned on 80 different views (viewpoint and illumination varied) of the texture, each associated with a different class under the model (and having class-specific hidden unit biases). Although the samples can be easily differentiated from the raw data, the directionality property is rather well captured with the model. Figure 4.14 shows texture interpolation results, with horizontal morphing between left and right source/target textures. Similar to earlier results, we can see the source/target textures being generated at their respective locations, and other kinds of texturing in the middle, with visually effective transitioning throughout the morph.

Future work for the texture morphing includes the development of joint models of texture data and (bias) control parameters.

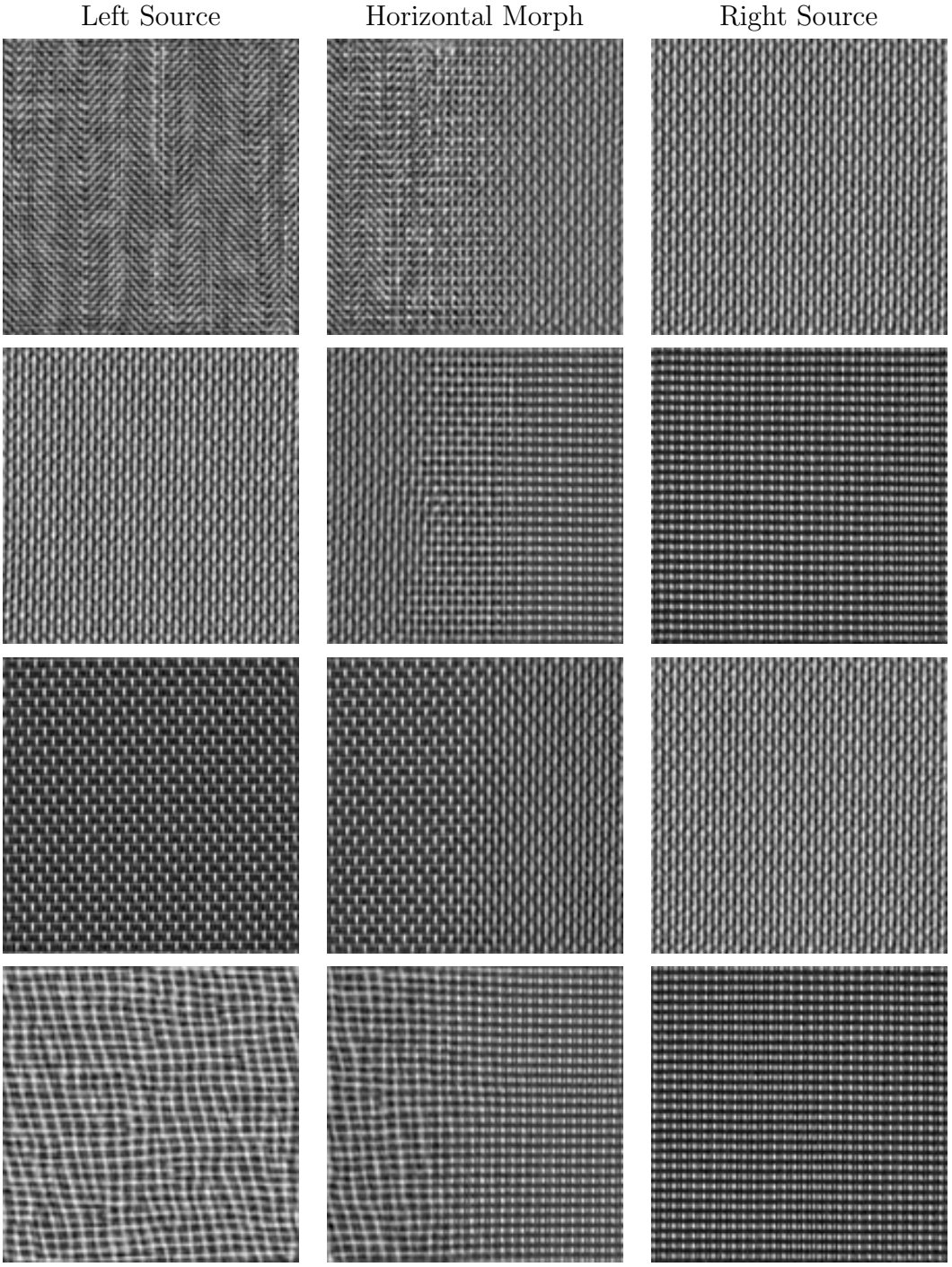


Figure 4.10: Example texture synthesis results obtained by a tiled convolutional Gaussian Bernoulli RBM (Tm) with weights of the multi-Tm of [1] with 256 features per site, trained on eight Brodatz textures, and biases set by convex combinations of two training-texture-specific biases. The left and right columns show samples by using training-texture-specific biases. The middle column shows samples by using bias fields whose biases are horizontally varied from one of the training-texture-specific biases to the other one. Samples using only the corresponding training texture specific biases are shown on the left and on the right of each case.

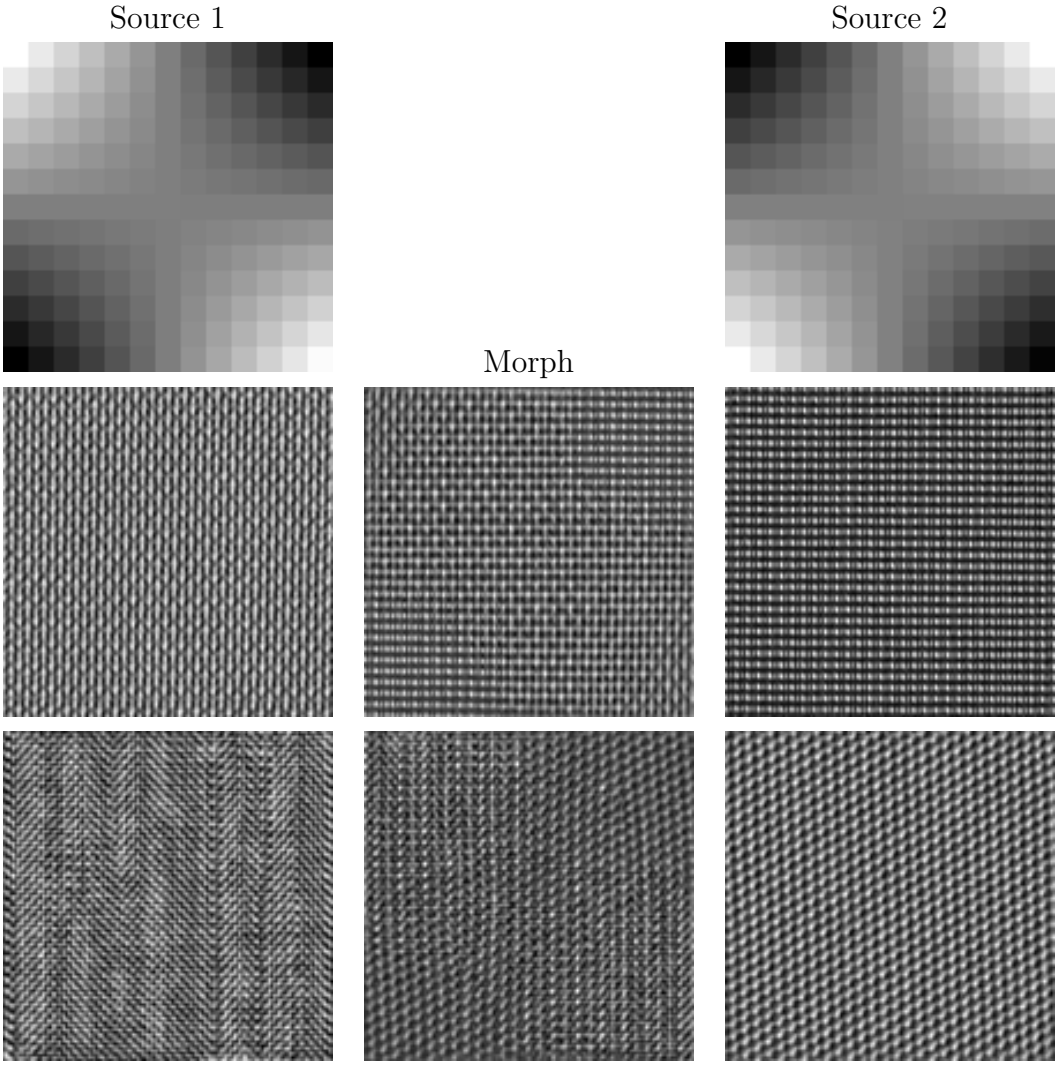


Figure 4.11: Example texture synthesis results obtained by a tiled convolutional Gaussian Bernoulli RBM (Tm) with weights of the multi-Tm of [1] with 256 features per site, trained on eight Brodatz textures, and biases set by convex combinations of two training-texture-specific biases. The middle column shows samples using bias fields in which the biases vary spatially. Specifically the training-texture-specific biases have been placed at the corners of the bias fields such that the biases of each pair of neighboring corners are different from each other (and so the biases are the same for the opposing corners). The biases at other locations are obtained by bilinear interpolation from these corner biases. The images on the left and on the right of each morph are samples by using the training-texture-specific biases. The top row visualizes the mixing proportions of the biases used in the morphing.

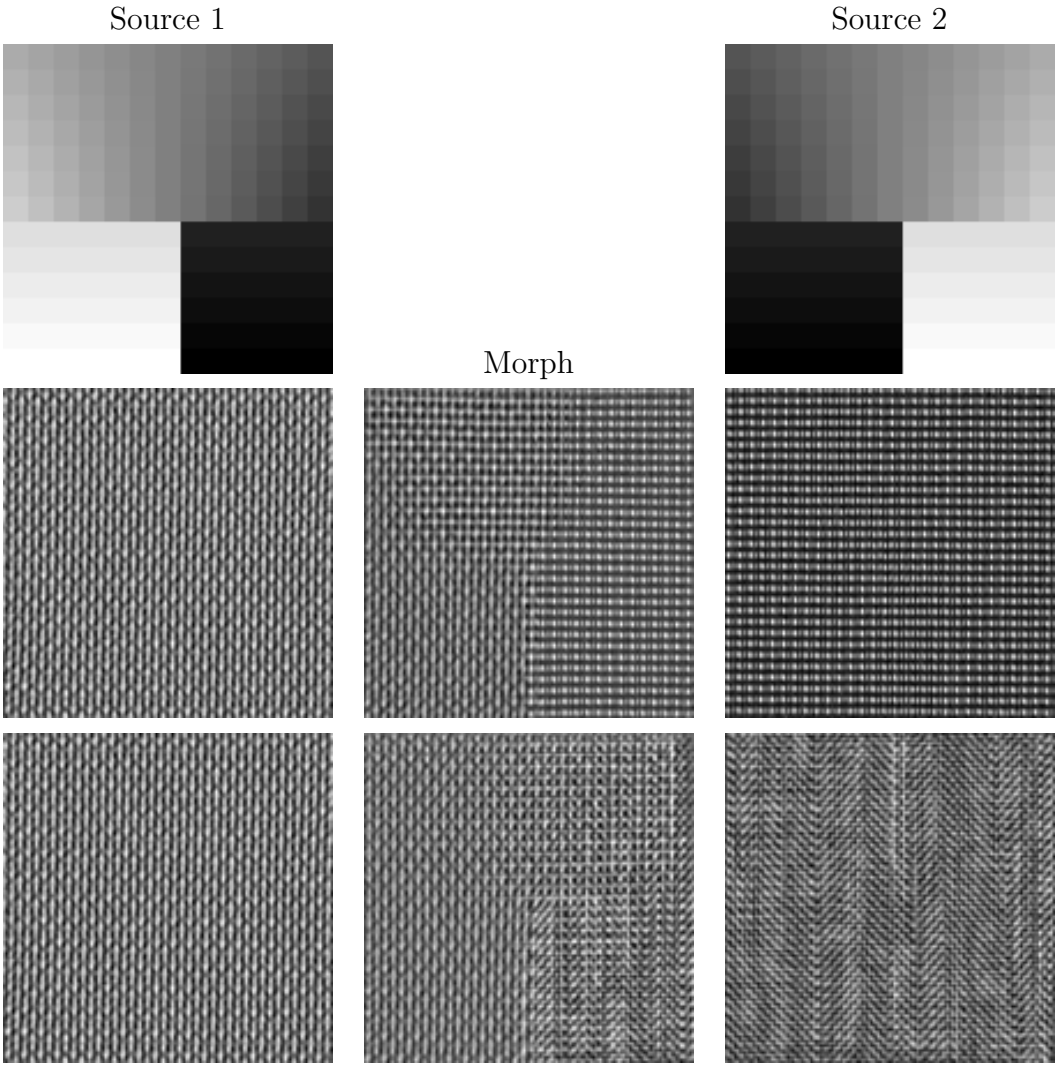


Figure 4.12: Example texture synthesis results obtained by a tiled convolutional Gaussian Bernoulli RBM (Tm) with weights of the multi-Tm of [1] with 256 features per site, trained on eight Brodatz textures, and biases set by convex combinations of two training-texture-specific biases. The middle column shows samples using bias fields in which the biases vary spatially. The images on the left and on the right of each morph are samples by using the training-texture-specific biases. The mixing proportions of the texture-specific-biases are visualized at the top row.

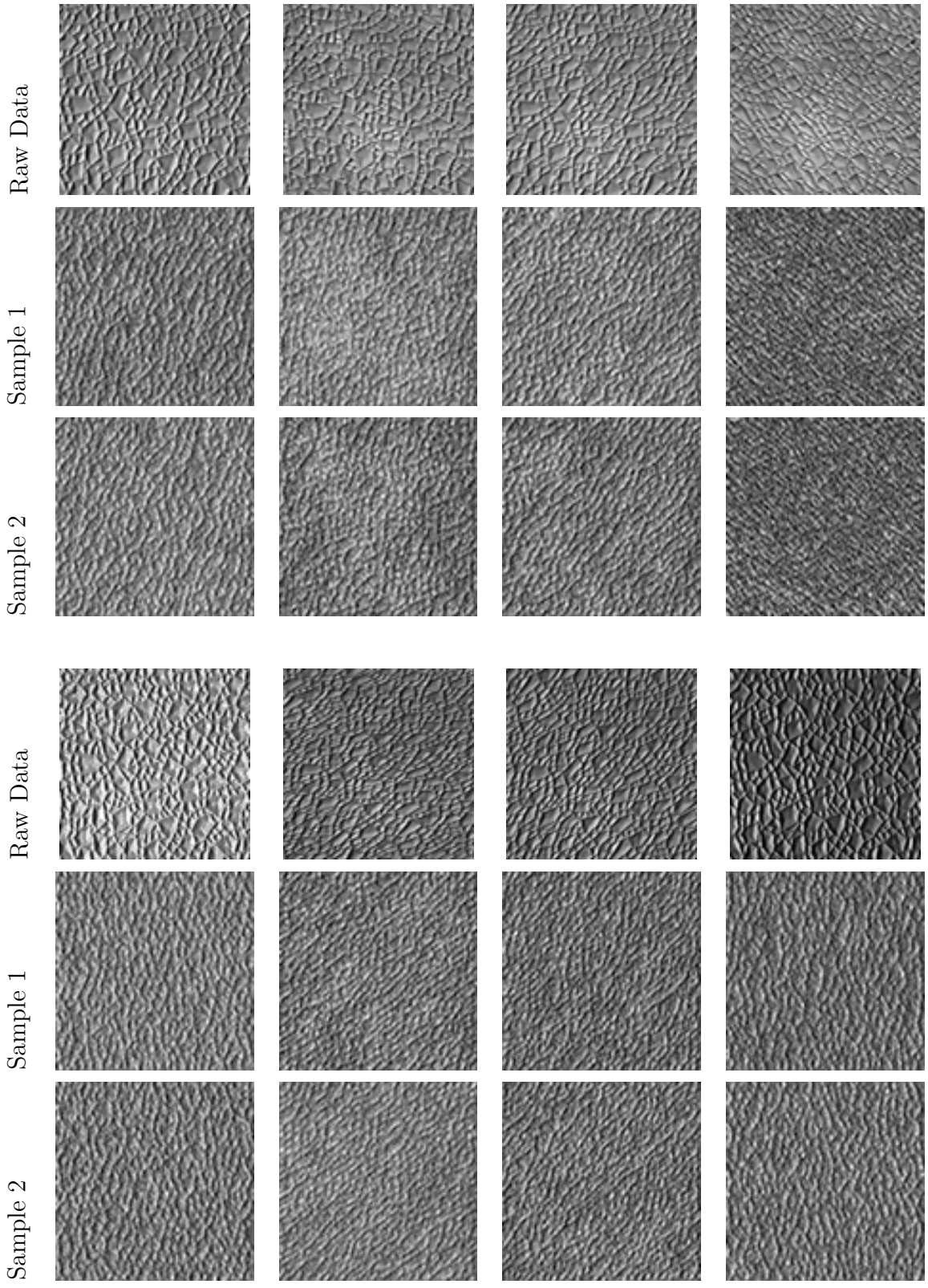


Figure 4.13: CUReT texture training views vs. model samples.

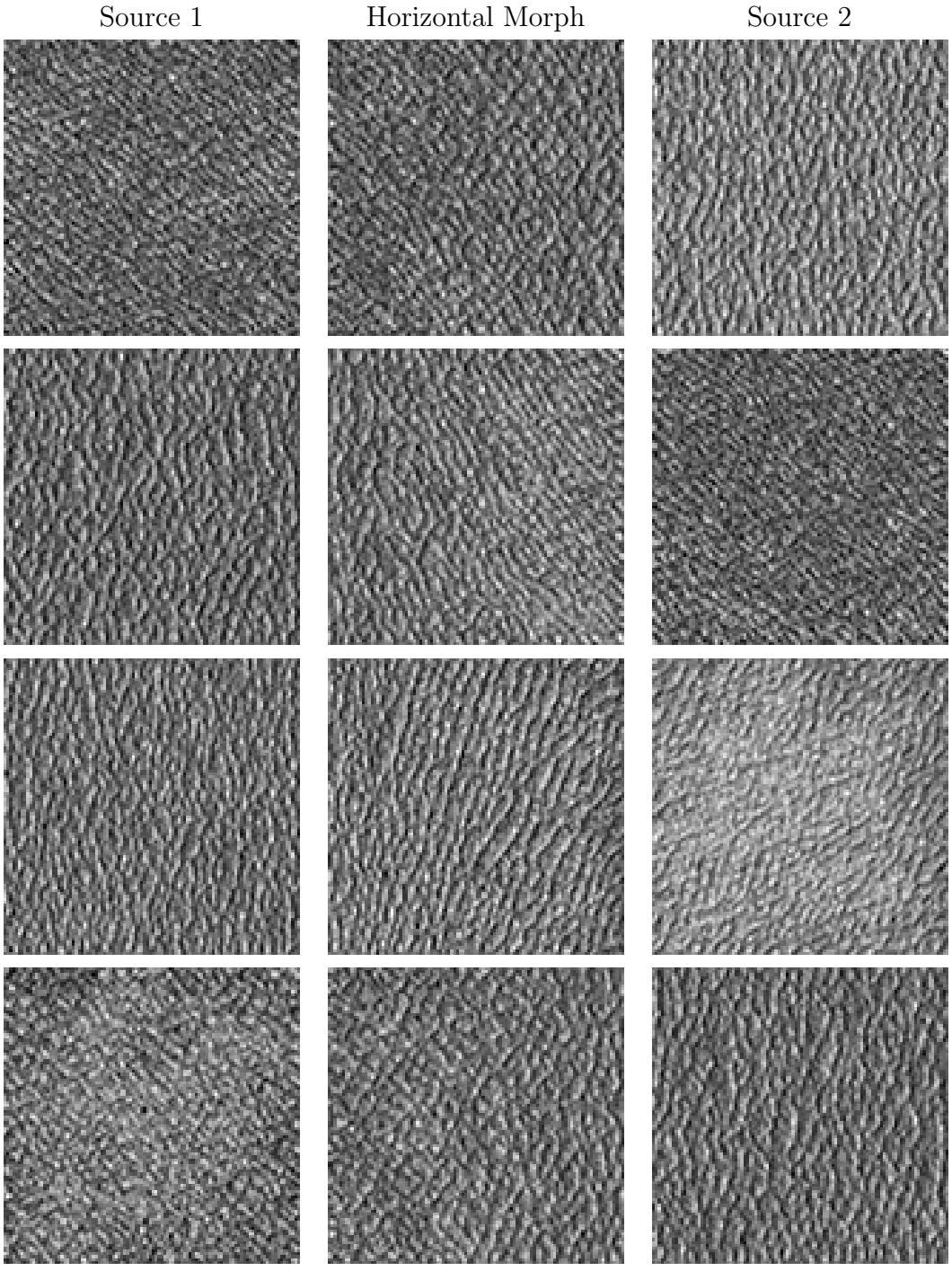


Figure 4.14: Interpolating horizontally between two different view samples of a CUReT texture.

4.6 Summary and Discussion

We have analyzed the generative power of mPoT and its subcomponents for the task of single texture modelling. Our results show that it is essential to not restrict the conditional mean of visible units to be zero, consistent with previous findings in Heess et al. [2009]. The results on the texture synthesis and inpainting tasks are generally as good as and sometimes better than the start-of-the-art results in Heess et al. [2009]. Our results show that for this task it is the mean hidden units that are much more important, especially in unconstrained sampling.

We have then developed Boltzmann machines capable of modelling multiple textures, and applied it to the tiled-convolutional Gaussian-Bernoulli RBM. By considering a shared set of weights but texture-specific hidden unit biases, we have shown comparable performance to the individually-trained texture models which already provide state-of-the-art results. The feature sharing by multi-texture models is expected to yield savings in terms of the number of features needed to model several categories, and provides a natural route for extension to a more comprehensive natural image model.

In Appendix C.1 we have also analyzed the effect on texture synthesis quality of using either hidden units and using block-Gibbs sampling or integrating them out and using HMC in model learning and in inference. We have there also developed novel boundary handling methods, assessed their effect on texture synthesis quality under the Tm-model, and discussed several other considerations for effective texture modelling.

We have demonstrated that the multiple texture Boltzmann machine based on the tiled-convolutional RBM can produce images containing multiple textures, with controlled spatial extent. We have then developed a texture interpolation method based on the multiple texture Boltzmann machine, and shown that by interpolating the hidden unit biases associated with training texture classes, novel textures can be generated. Textures can be interpolated spatially by simply interpolating the biases spatially.

Future work includes extending the model to be able to switch (without supervision) between generating differently textured regions, and interpolating them. This can be done by letting the *biases* $\mathbf{b}$ in $E_{\mathrm{GB-RBM}}$ (or the bias combination weights in 4.5) depend on a higher layer of hidden units.

Chapter 5

Multistream Networks for Visual Boundary Prediction

5.1 Introduction

In this chapter we consider predicting visual boundaries in natural images. Martin et al. [2004] give the definition "A boundary is a contour in the image plane that represents a change in pixel ownership from one object or surface to another." Boundary detection is to be contrasted with edge detection, which is a low-level technique to detect an abrupt change in some image feature such as brightness or colour. So, for example, a heavily textured region might give rise to many edges, but there should be no boundary defined within the region. Thus it is more involved with detecting abrupt changes in more global information, such as texture. Accurately finding boundaries subserves many vision tasks including segmentation, recognition and scene understanding.

Recent work on boundary detection makes heavy use of the ground truth provided by the Berkeley Segmentation Data Set (BSDS) [Arbelaez et al., 2011], where each of the 500 images was processed by multiple human annotators. The (deliberately vague) instructions to the annotators were [Martin et al., 2004]: "Divide the image into some number of segments, where the segments represent 'things', or 'parts of things' in the scene. The number of segments is up to you, as it depends on the image. Something between 2 and 30 is likely to be appropriate. It is important that all of the segments have approximately equal importance."

Our main interest in the thesis has been building generative models of natural image data based on Boltzmann machines. Generation of texture is clearly a

necessary subcomponent of any credible model for visual scenes. So also is the creation of several image segments, and thus the existence of contours as boundaries between segments. As we saw in the previous chapter focusing on modelling textures based on conditionally Gaussian models, it is important to have hidden units to control their means (especially for unconstrained synthesis). While we showed that multiple textures can be created based on 'mean-only' models, results in the literature suggest the importance of covariance units for modelling the more generic class of natural images.

Our goal in this chapter is two-fold: Firstly, we wish to improve understanding of what are the roles of the mean and the covariance hidden units of such conditionally Gaussian models in the generation of image segment boundaries. Secondly, we investigate deep neural network architectures for learning the boundary detection problem, which is expected to be useful also for further generative model development.

Notable aspects of the work is (i) the use of "covariance features" [Ranzato and Hinton, 2010] which depend on the *squared* response of a filter to the input image, and (ii) the integration of image information from multiple scales and semantic levels via multiple streams of interlinked, layered, and non-linear "deep" processing in an end-to-end optimizable architecture. Our results on the Berkeley Segmentation Data Set 500 (BSDS500) show comparable or better performance to the top-performing methods gPb [Arbelaez et al., 2011], SCG [Ren and Bo, 2012], and Sketch Tokens [Lim et al., 2013]. Additionally, our approach provides the fastest reported prediction times, and avoids several hand-engineered and/or computationally complex designs which are part of the the first two approaches cited above. We provide a careful disection of the performance in terms of architecture, feature-types used and training methods, which provides clear signals for model understanding and further development.

The structure of the chapter is as follows: the following section contains the theory part, including descriptions of the models considered and related work. Section 5.3 describes the experiments, including descriptions of the data considered, model training, and inference results. Section 5.4 provides a summary, and describes ongoing and future work[1].

[1]I thank Nicolas Heess for feedback, comments and discussions on many parts of the work presented in this chapter. His suggestions concerning the presentation of Kivinen et al. [2013b] have also been useful here. The main ideas and work presented in this chapter are by the thesis author, supervised by Prof. Christopher K. I. Williams.

5.2 Theory

In section 5.2.1 we describe a variation of the mean-and-covariance restricted Boltzmann machine (mcRBM) architecture of Ranzato and Hinton [2010], and its deep belief net extension, both of which we have developed and which we use as basic feature extraction models. The use of these features in supervised learning of boundary prediction is described in section 5.2.2, and related work is discussed in section 5.2.3 .

5.2.1 The mcRBM and a mcDBN extension

The mcRBM-model [Ranzato and Hinton, 2010] (see also Sec. 2.1.2.7) is a generative model for images. We assume the following generalization of the model so that it assigns an energy to the joint configuration of visible units $\mathbf{v}$ and its hidden units $\mathbf{h}$ as follows:

$$E = -\sum_j h_j^c \left(d_j - \sum_f \frac{\pi_{fj}}{2} \left[\mathbf{K}_{\cdot f}{}^\top \mathbf{A}\mathbf{v} \right]^2 \right) + \sum_i \frac{(v_i - a)^2}{2\sigma^2} - \sum_\ell h_\ell^m \left(b_\ell + \frac{1}{\sigma^2} \mathbf{M}_{\cdot \ell}{}^\top \mathbf{v} \right), \tag{5.1}$$

where $\mathbf{K}_{\cdot f}$ denotes a factor-to-image-units filter for a factor with index f, and we define the pooling matrix elements so that $\pi_{jf} \geq 0 \ \forall \ j, f$ and $\sum_f \pi_{jf} = 1 \ \forall \ j$. Extending the basic model, we introduce/consider $\mathbf{A}$, which denotes a whitening basis[2]. $\mathbf{M}_{\cdot \ell}$ denotes the mean-hidden-unit-to-visible-unit filter for unit type ℓ. Each of the covariance units h_j^c and mean hidden units h_ℓ^m are associated with biases d_j and b_ℓ, respectively. a is the visible unit bias, and σ is a positive scalar. The joint hidden unit probability distribution conditional on the visibles factorizes over sites:

$$p\big(h_j^c = 1 \mid \mathbf{v}, d, \pi, \mathbf{K}\big) = \text{sig}\left(d_j - \frac{1}{2} \sum_f \pi_{fj} \left[\mathbf{K}_{\cdot f}{}^\top \mathbf{A}\mathbf{v} \right]^2 \right), \tag{5.2}$$

$$p(h_\ell^m = 1 \mid \mathbf{v}, b, \sigma, \mathbf{M}) = \text{sig}\left(b_\ell + \frac{1}{\sigma^2} \mathbf{M}_{\cdot \ell}{}^\top \mathbf{v} \right), \tag{5.3}$$

where $\text{sig}(z)$ denotes the logistic sigmoid function $\text{sig}(z) = 1/(1 + e^{-z})$.

[2]The basis was learned from all 8×8 patches in the training data. Each of the 63 retained dimensions was scaled according to the inverse of the square root of the associated eigenvalue of the scatter matrix, similar to ZCA. A whitening front-end for covariance filters has been used also in the TmPoT in Ranzato et al. [2010b], and in our experiments with such models in Chapter 4. We note that it is unclear from Ranzato et al. [2010b, page 6, paragraph 1] whether the above variant has been explored, as the whitening-frontend footnote refers to TmPoT code only (no code for the TmcRBM, or comments on such is provided).

The models used in our experiments assume diagonally-tiled convolutional parameter sharing, with 8×8 receptive fields, stride of 2 units, and 64 features of both kinds per site. Figure 5.1 illustrates a diagonally-tiled convolutional mcRBM (TmcRBM) model instance (with different receptive field sizes and stride than what are used in our experiments). We fixed the pooling matrix π to the identity, σ to unity, and the visible unit bias to zero. Each full image to be processed with the model was normalized to have zero mean and unit variance.

Figure 5.2 (left) shows samples from a generatively trained TmcRBM, which have been drawn with HMC, and from a tiled convolutional mPoT (TmPoT) with pooled covariance filter-outputs (a state-of-the-art shallow image model TmHPoT) [Ranzato et al., 2010b] are shown in Figure 5.2 (right). Qualitatively the samples are similar to those shown in Ranzato et al. [2010b, Fig. 3, panel C].

Figures 5.3 and 5.4 visualize hidden unit activations as a response to different image patches under the TmcRBM. In each of these figures, the rectangles overlaid on the input image denote boundaries of receptive fields of example hidden unit stacks, for each of the four different parameter sets (due to 8×8 filters and a stride of 2). Together the stacks form all of the hidden units which connect to the visible unit position marked with a magenta-coloured circle. The figures also visualize the weights of these sets, and the activations of the units at the positions. We can see that many of the covariance filters of the model are very clearly structured, and resemble Gabor-filters; in contrast the mean hidden unit filters are less localized in general.

Covariance hidden units of the kind that are on penalize edge-like structure at the corresponding location, scale and orientation. Intuitively the covariance units are expected to want to be off at such edge-like image structures, as otherwise there would be a large energy increase, due to the squared response of the filter and the matched edge (either positively or negatively correlated). Using covariance units for structures having invariance to the sign of the input is more economical than when using the mean units, which are sensitive to the sign. In natural images edge-like structures appear often in their sign-inverted form (dark-to-light versus light-to-dark edge), although there exist certain biases due e.g. to gravitation affecting the orientation of structures, and also the direction of illumination.

In Figure 5.3 the receptive fields are located at a vertical surface boundary (between a building and sky). Notice that these stimuli give rise to very small

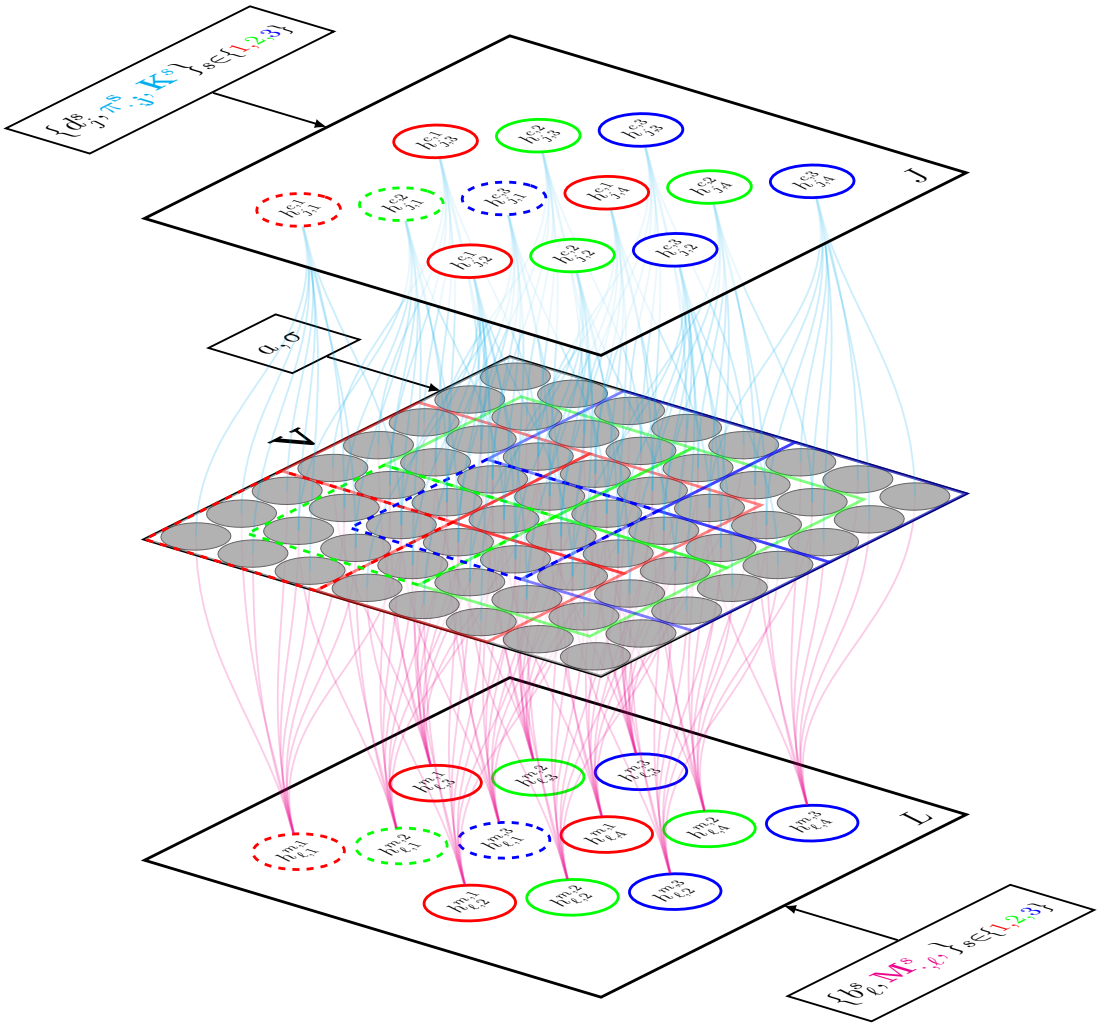


Figure 5.1: A graphical illustration of a diagonally-tiled-convolutional mcRBM. The visible units visualized in the middle layer connect to the J covariance hidden unit layers of the model at the top, and to the L mean hidden unit layers of the model at the bottom. Within these layers the hidden units are partitioned into different sets (red,green,blue), associated with different parameters for their hidden units. Each of the hidden units connect to a region of visible units, and the filter applications within a set are non-overlapping and tile a certain-sized region of visible units, and those of different sets are offset diagonally with a stride between the neighboring sets. .

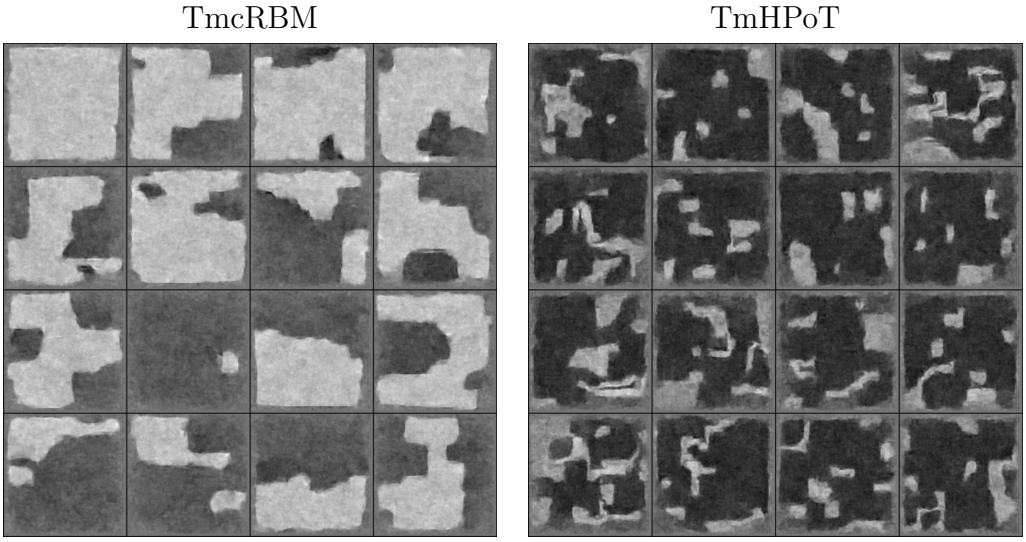


Figure 5.2: Samples (130×130 boundary-cropped images from 142×142 samples) from a TmcRBM (left), and from a TmHPoT [Ranzato et al., 2010b] (right; from the authors), trained on 200 training images from the BSDS500. The sample images are globally normalized to fill the full intensity range.

activations (very dark blocks) for several covariance units, and that their filters have mostly-vertical Gabor-type appearance. The other covariance units have much larger activations, and which are similar to each other. Some of the units have filters with mostly-vertical Gabor-type appearances, but for example are not centered at the boundary. In Figure 5.4 the receptive fields have sharp intensity changes in several directions, and much larger amount of covariance-unit inactivation can be observed.

We develop a deep belief network (called the mcDBN) from the mcRBM, extending it to have an additional layer of binary hidden units on top, similar to the DBN extension of the mPoT model in Ranzato et al. [2011a]. We also use diagonally-tiled convolutional feature sharing there, using a stride of one second layer unit (corresponding to 8 units in the visible layer). A top-layer hidden unit layer takes input from a 3×3 region of mcRBM $(64 + 64)$ hidden unit stacks under each of the 4 shifts, containing 512 input 'channels' in total. Thus each of the second-layer hidden units are directly influenced by a 30×30-region of visible units[3]. We used 512 feature planes, and thus the second-layer has 3 sets of 512 hidden units, each with their own sets of weights and biases. Figure 5.5

[3] 3 replicas of 8×8 filters at all of the diagonal-2-shifts (6 unit offsets between the 1^{st} and the last).

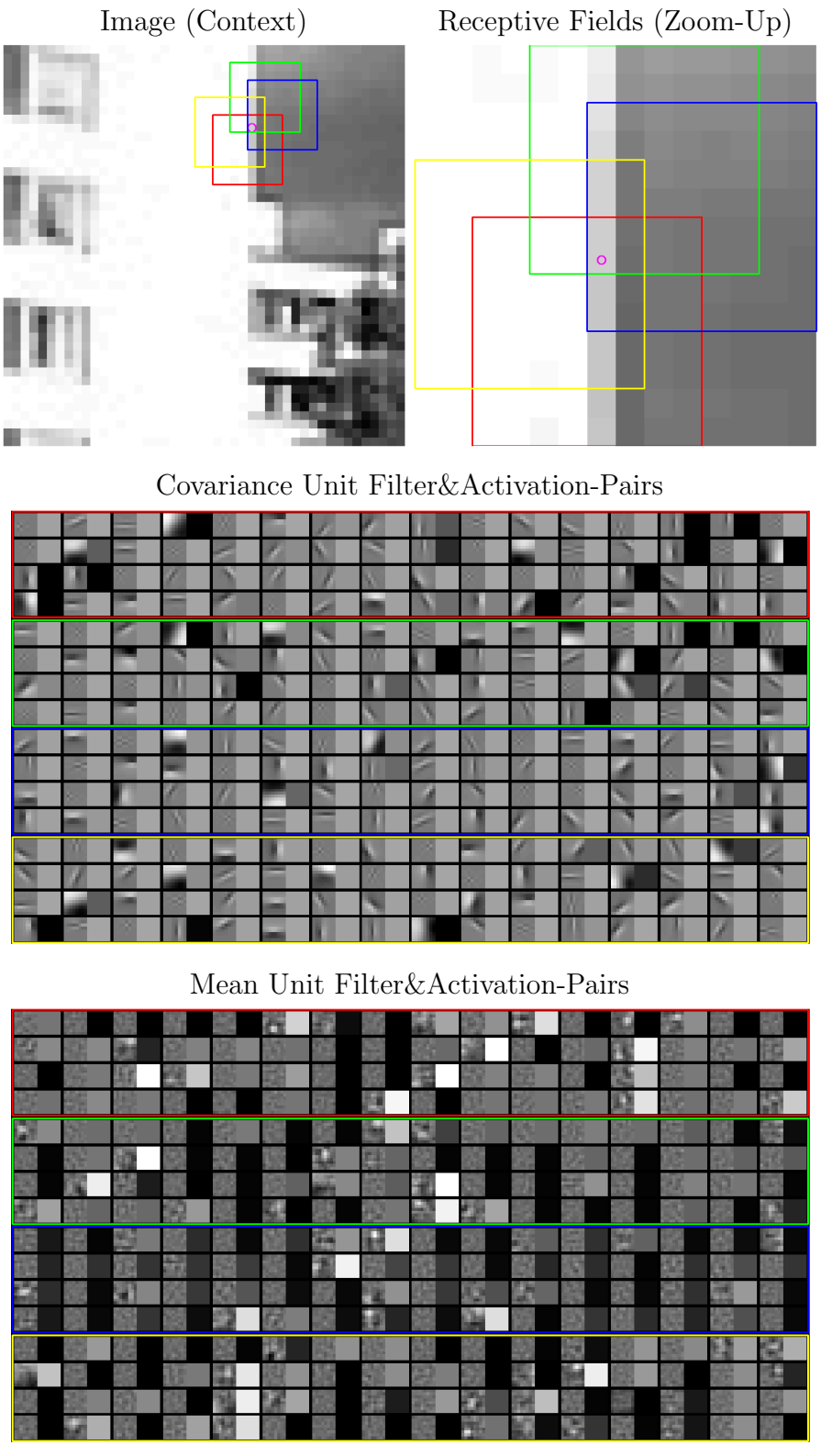


Figure 5.3: Hidden unit activations as a response to image patches under a TmcRBM. The rectangles in the input image denote boundaries of receptive fields of example hidden unit stacks, under each of the four different parameter sets. Together the stacks form all of the hidden units which connect to the visible unit position marked with a magenta-coloured circle. The weights of these sets, and the activations of the units at the positions are visualized in horizontal pairs, with a pair per black rectangle. White/black activation means an active/inactive feature. Best viewed on screen.

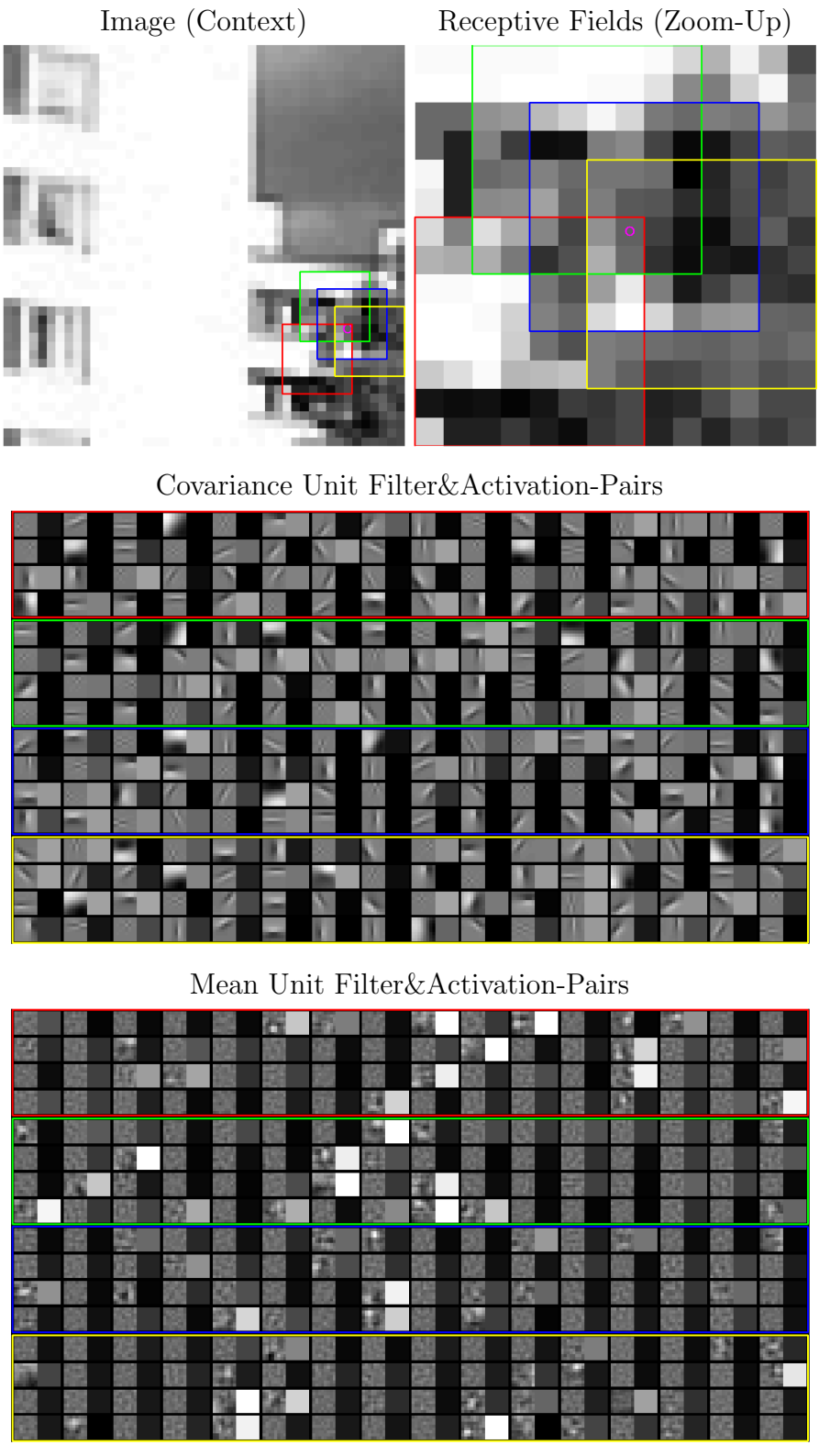


Figure 5.4: Hidden unit activations as a response to image patches under a TmcRBM. The rectangles in the input image denote boundaries of receptive fields of example hidden unit stacks, under each of the four different parameter sets. Together the stacks form all of the hidden units which connect to the visible unit position marked with a magenta-coloured circle. The weights of these sets, and the activations of the units at the positions are visualized in horizontal pairs, with a pair per black rectangle. White/black activation means an active/inactive feature. Best viewed on screen.

illustrates a diagonally-tiled convolutional mcDBN (TmcDBN) model instance. Note that in the illustration the receptive fields are much smaller than in the model considered in our experiments, and the connections are drawn only from few selected hidden units to their receptive fields.

5.2.2 Supervised boundary prediction

We consider feedforward neural networks for boundary prediction, where each pixel v_i has a corresponding contour unit. Let u_i denote the predicted probability that contour unit i takes value 1 (i.e. is a contour). Based on a vector of features $\mathbf{x}$ (which could arise from multiple hidden layers of a network), the prediction u_i is obtained via logistic regression according to:

$$u_i = \text{sig}\left(g + \sum_j W_{ji} x_j\right), \tag{5.4}$$

where W_{ij} denotes a weight between feature x_j and contour unit site i, and g denotes a scalar bias. We are considering many networks with several layers of logistic units, starting from the initial features defined by the activation probabilities of the mean and covariance units, as in (5.3) and (5.2), respectively.

We consider three architectures of boundary prediction networks, as shown in Fig. 5.6. The "shallow" network has only a single layer of hidden units, those corresponding to the features of the mcRBM model. The "deep stream" architecture makes contour predictions based on the mcDBN-type hidden units, while the "two-stream" architecture (see Fig. 5.6 and 5.7) uses the connection patterns of both the shallow and deep streams via skip-layer connections (see e.g. Ripley [1996, page 144]). Importantly we can think of each stream having two parts: image feature extraction, and hypothesis propagation/read out (solid vs. dashed lines in Fig. 5.6)[4]. In our networks, we use mirrored connectivity structure for these two parts. For example, in the shallow networks each hidden unit receives input from an 8×8 region of visible units, and sends information to an 8×8 region of contour units, with the same relative positioning. Adding an additional encoding layer thus adds another read-out hidden layer. One reason for doing so is the reduction in the hidden unit grid locations in the deeper networks. For example for a 142×142 input image, there are only 5×5 hidden unit stack positions

[4]The boundary between these can be blurred in practice, especially in the case that network-wide parameter changes are allowed in training.

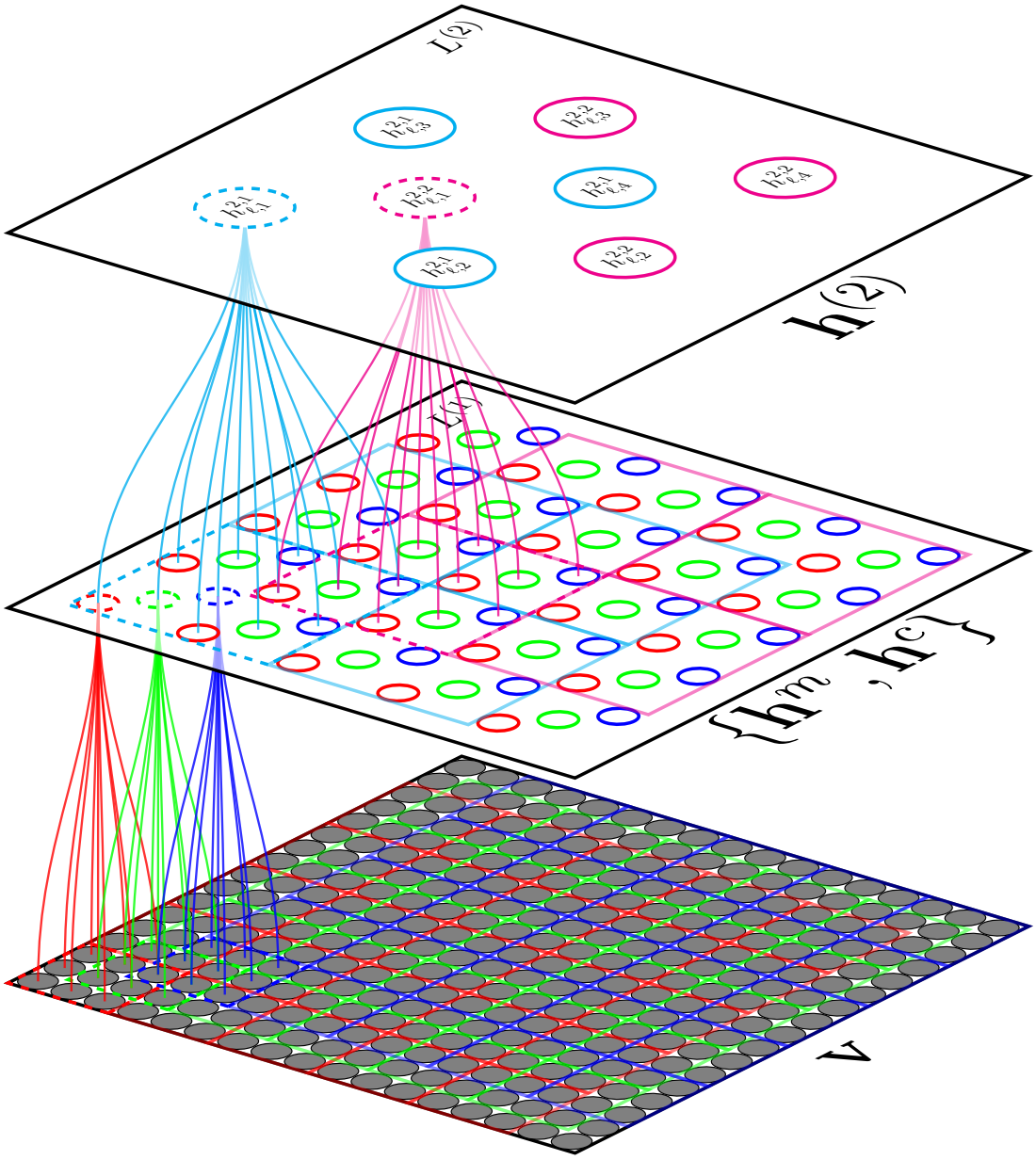


Figure 5.5: A graphical illustration of a diagonally-tiled-convolutional mcDBN. The visible units $\mathbf{v}$ visualized in the bottom layer connect to the covariance hidden unit $\mathbf{h}^c$ and mean hidden unit $\mathbf{h}^m$ layers ($L^{(1)}$ feature planes in total) in the middle, which connect to the second layer hidden units at the top (having $L^{(2)}$ feature planes). Within the hidden unit layers, the units are partitioned into different sets (red,green, and blue in the first layer, and cyan and magenta in the second layer), associated with different parameters for their hidden units. Each of the hidden units connect to a region of units below, and the filter applications within a set are non-overlapping and tile a certain-sized region of units, and those of different sets are offset diagonally with a stride between the neighboring sets. Connections are drawn only from few selected hidden units to their receptive fields.

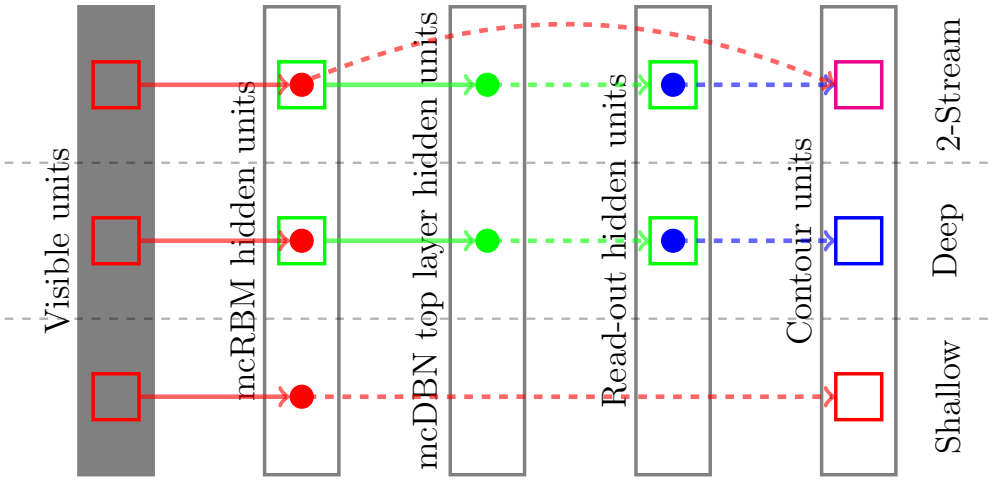


Figure 5.6: An illustration of the different streams in the considered networks. The dots in the layers illustrate hidden units, and the blocks receptive fields (with no size-consistency). The solid and dashed arrows denote feature extraction parameters, and hypothesis read-out parameters, respectively.

at the second hidden layer, per each of the 3 shifts. In these deeper streams, the number of feature planes is also applied in an anti-symmetrical fashion, and so the first and the last hidden layers in the stream have equal numbers of planes. More complicated structures could be considered, such as using (additional) hidden layer(s) in the integration of the streams, with possible skip-layer connections for gradual integration.

Figures 5.8, and 5.9 visualize contour prediction for an input pattern with a shallow model ([mcRBM init] using the notation of Sec. 5.3.3.1), showing the covariance hidden unit feature activation and contribution patterns in Figure 5.8, and those of the mean hidden units ones in Figure 5.9. Figures 5.10, and 5.11 show similar analysis, but for a different input pattern. The prediction messages in the figures are relative contributions of the particular features to the contour unit input (hidden unit activation times its weight to the contour unit, normalized (by maximum absolute value) across all hidden units in the receptive field for the particular contour unit site denoted by the magenta circle). In the case of Figures 5.8 and 5.9, the receptive fields contain a surface boundary, whereas in the case of Figures 5.10 and 5.11 there is no boundary. We can see that in the former case, the covariance feature contribution to the prediction is clearly stronger.

The models containing multiple streams are motivated by the need to capture image information at multiple scales and semantic levels. The shallow networks

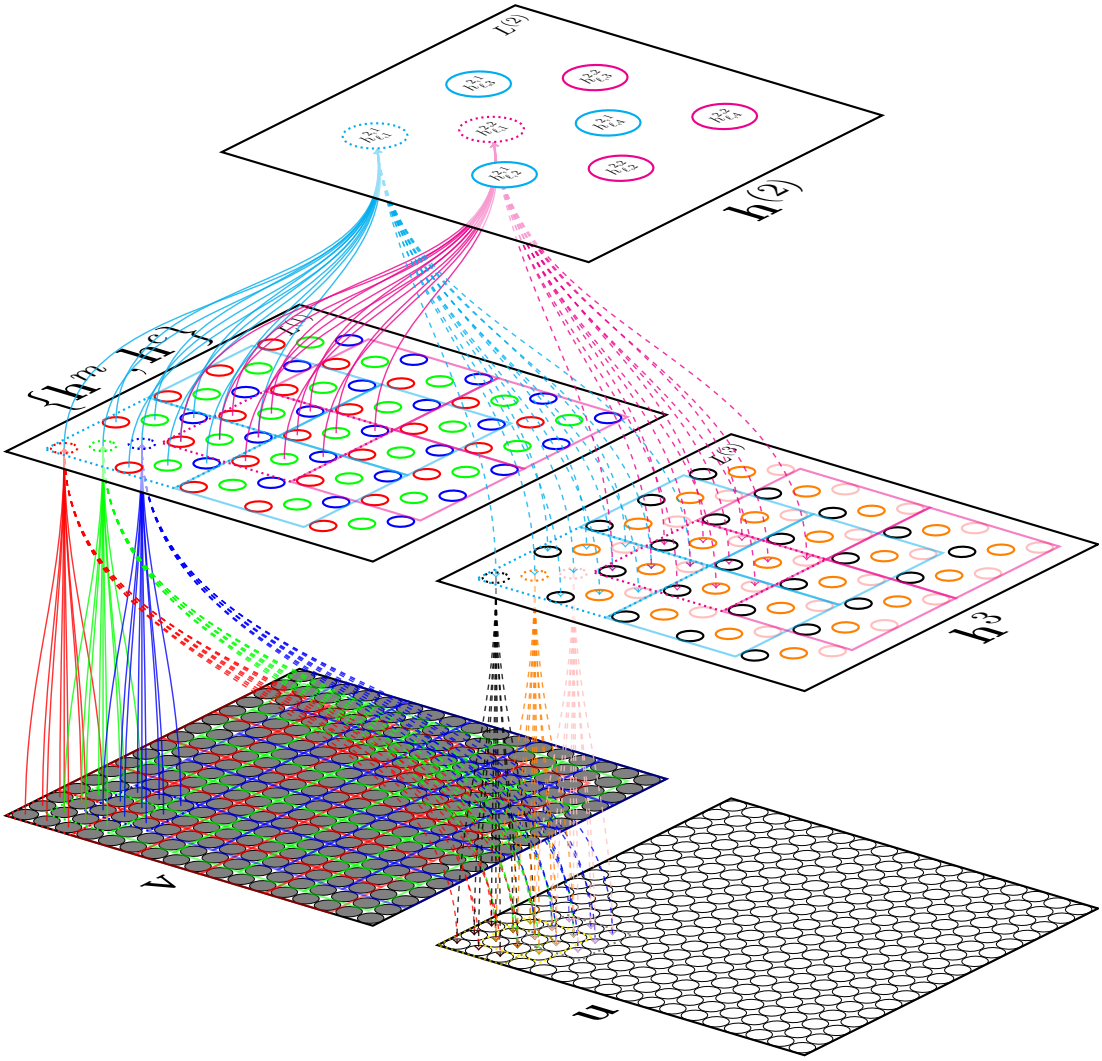


Figure 5.7: A graphical illustration of a two-stream model for contour prediction. The visible units $\mathbf{v}$ in bottom left send information to the covariance and mean hidden units directly above, which send information to the second layer hidden units at the top and also to the contour units $\mathbf{u}$ at the bottom right. The second layer hidden units send information to the hidden units $\mathbf{h}^3$ below right, which send information to the contour units below. As before, within the hidden unit layers, the units are partitioned into different sets, associated with different parameters for their hidden units. Receptive and send connections are shown only for few of the hidden units.

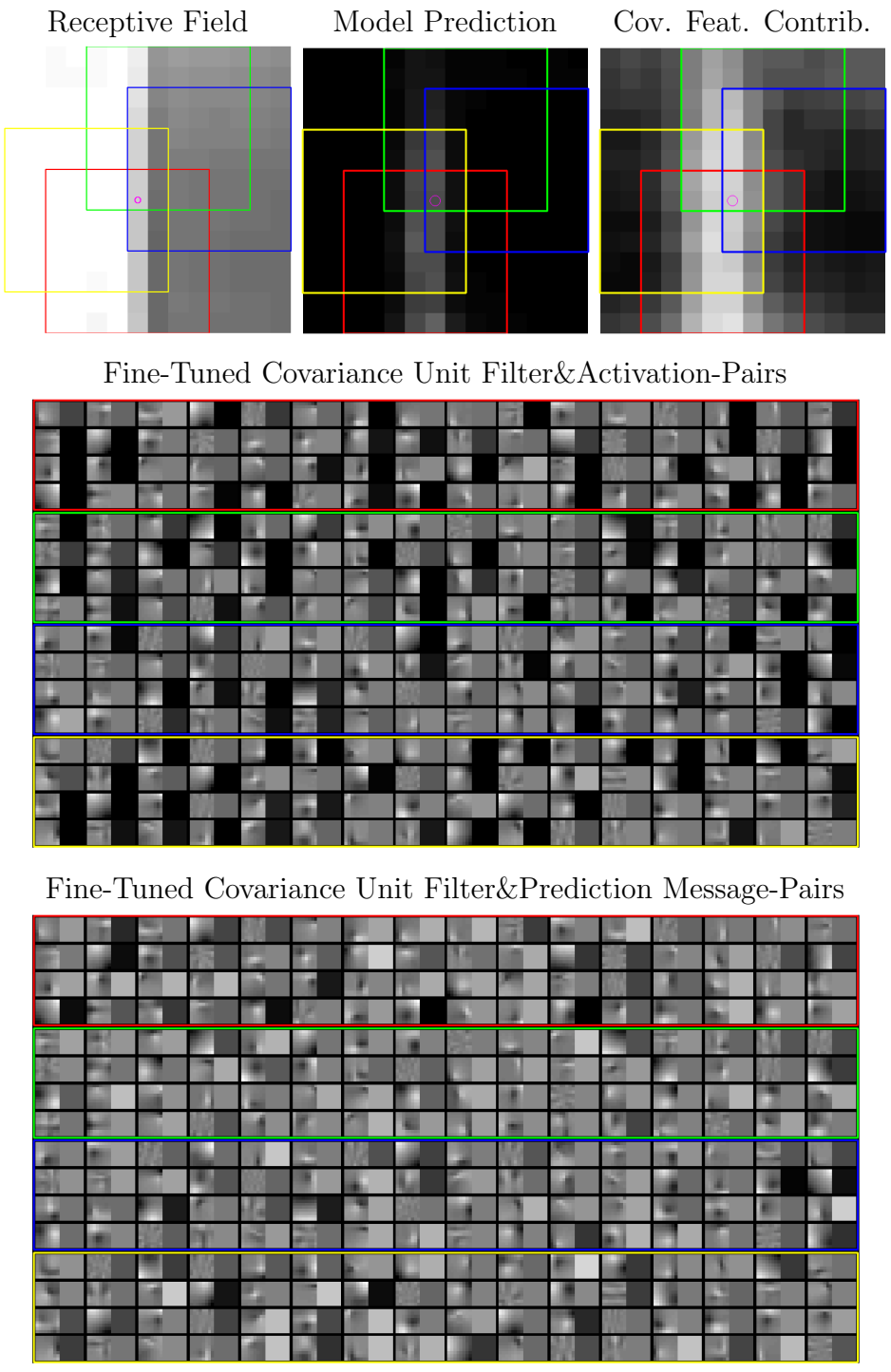


Figure 5.8: Contour prediction properties under a shallow full model [mcRBM init]. The covariance unit filters and activation pairs are visualized as e.g. in Fig. 5.4. The prediction messages are outputs of the features to the (input of the) particular contour unit denoted by the magenta circle. The model prediction (logistic function of contour unit input at each site) value is visualized with intensity, with white denoting probability of one for boundary, and black denoting probability of zero for boundary. Best viewed on screen.

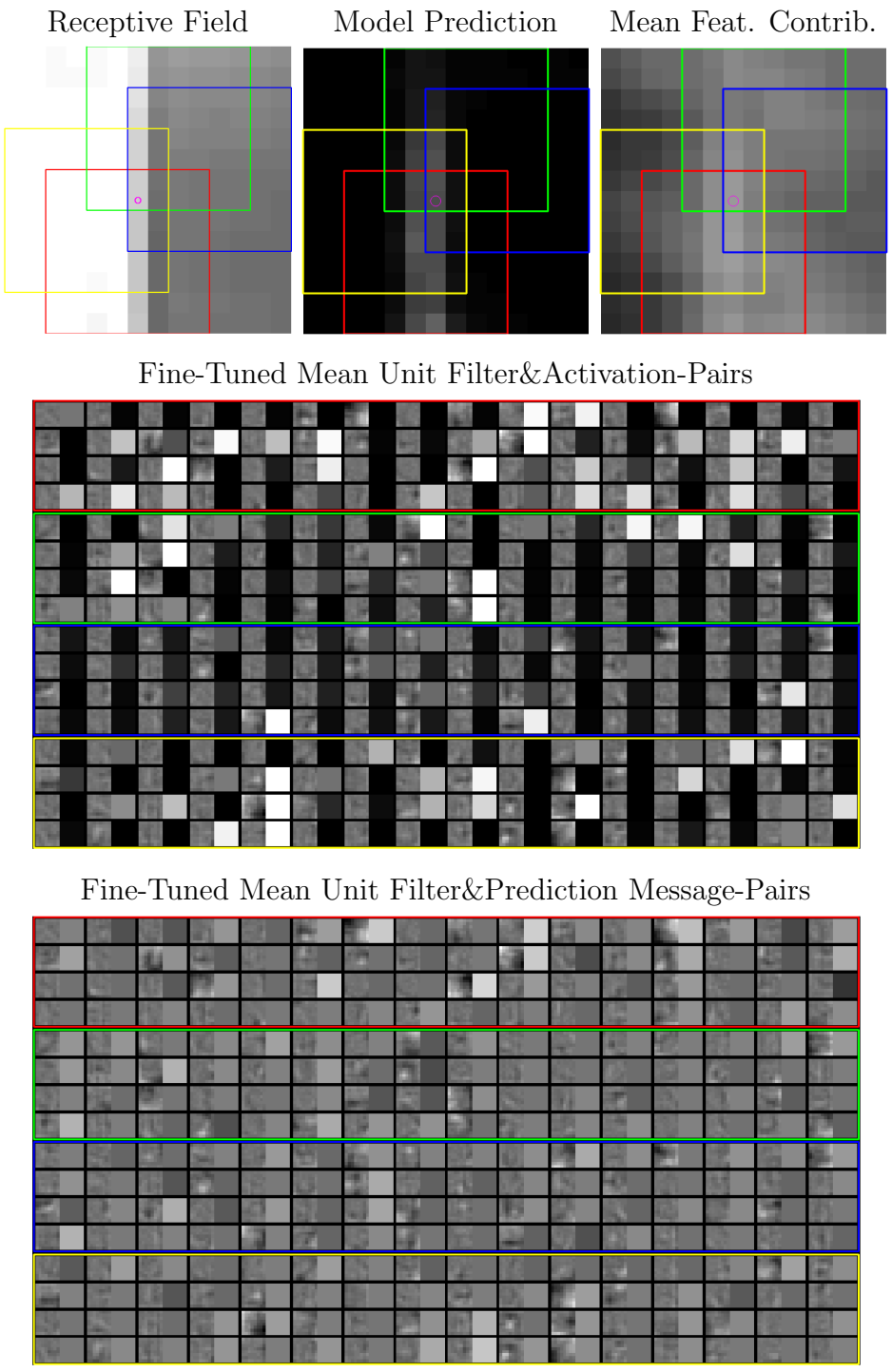


Figure 5.9: Contour prediction properties under a shallow full model [mcRBM init]. The mean unit filters and activation pairs are visualized as e.g. in Fig. 5.4. The prediction messages are outputs of the features to the (input of the) particular contour unit denoted by the magenta circle. The model prediction (logistic function of contour unit input at each site) value is visualized with intensity, with white denoting probability of one for boundary, and black denoting probability of zero for boundary. Best viewed on screen.

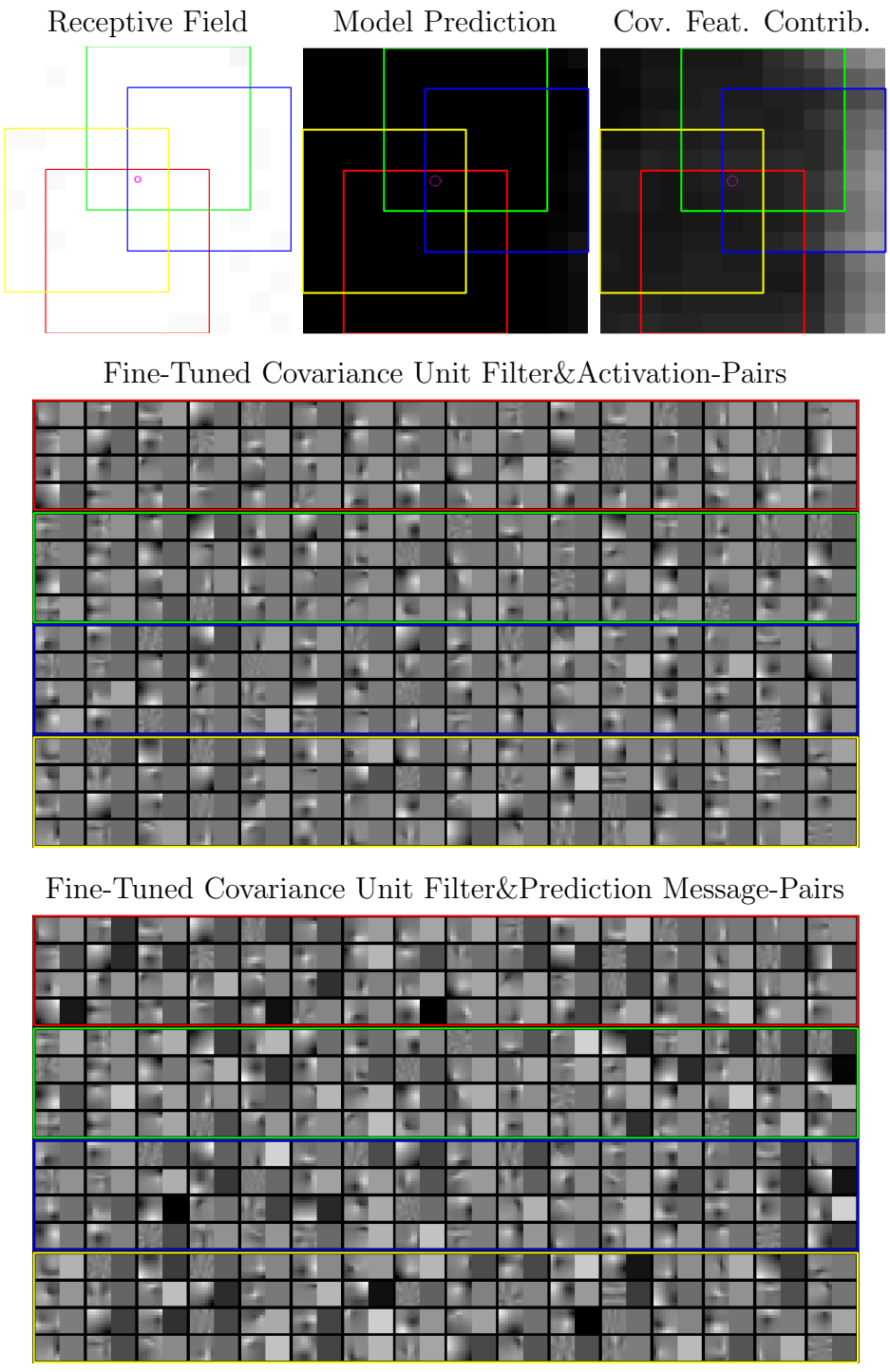


Figure 5.10: Contour prediction properties under a shallow full model [mcRBM init]. The covariance unit filters and activation pairs are visualized as e.g. in Fig. 5.4. The prediction messages are outputs of the features to the (input of the) particular contour unit denoted by the magenta circle. The model prediction (logistic function of contour unit input at each site) value is visualized with intensity, with white denoting probability of one for boundary, and black denoting probability of zero for boundary. Best viewed on screen.

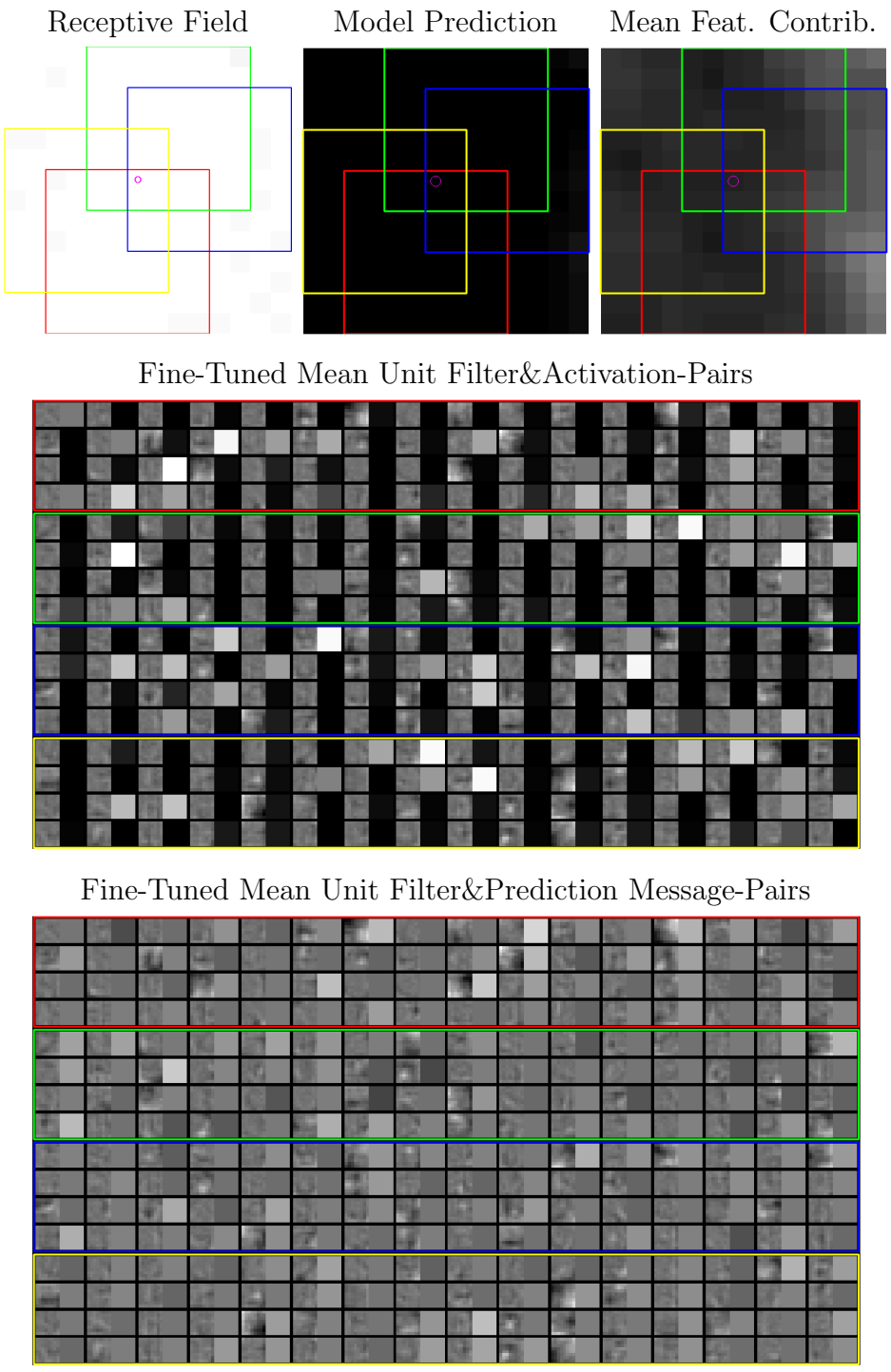


Figure 5.11: Contour prediction properties under a shallow full model [mcRBM init]. The mean unit filters and activation pairs are visualized as e.g. in Fig. 5.4. The prediction messages are outputs of the features to the (input of the) particular contour unit denoted by the magenta circle. The model prediction (logistic function of contour unit input at each site) value is visualized with intensity, with white denoting probability of one for boundary, and black denoting probability of zero for boundary. Best viewed on screen.

analyze the input image very locally, each site being affected by a 14×14 pixel area around it. It is expected that such a network could only detect very local image discontinuities such as edges. In order to distinguish texture discontinuities effectively, deeper networks taking input from larger areas are likely to be needed. Our experiments show significantly better results with such models.

Due to the sparse spatial application of the filters throughout the networks, the deeper scales are expected to have only much coarser-scale information about the data, and contour sites with very local discontinuities cannot be detected efficiently. Being able to consider combinations of the streams is expected to be beneficial, and our results for this demonstrate enhanced recall-rates over the deep-only networks.

5.2.3 Related work

The Canny edge detector [Canny, 1986] computes the edge response magnitude $\sqrt{G_x^2 + G_y^2}$ at each pixel, where G_x (resp. G_y) denotes the response of a Gaussian first derivative in the x (resp. y) direction, followed by stages of non-maximum suppression and hysteresis thresholding. Note that like our method it involves a squared filtering operation followed by non-linear processing, but in contrast there are a small, hand-crafted set of filters and post-processing steps.

An important reference method for boundary detection is gPb [Arbelaez et al., 2011]. gPb is based on a Pb (probability of boundary) [Martin et al., 2004] predictor that considers differences between histograms of brightness, colour and texture in two opposing half-circles, which together compose a circular receptive field, around a position at several orientations. The brightness and colour histograms are based on luminance and chrominance data (respectively) whereas the texture histograms are based on counts of vector quantized filter responses assigned to their nearest prototypes (textons). Second-order Savitsky Golay-filtering is used in conjunction of some of the features for improving localization of the contour detector, which operates on the half-circle feature difference computation results. Martin et al. [2004] study using Pb with several different detectors, finding logistic regression-based methods favorable, and demonstrate significantly better results with Pb than with the Canny edge detection method (which is the best out of several edge detection methods considered) on the BSDS300 data. In gPb, the difference computations are performed and combined across multiple scales

(using receptive fields of several sizes; also done in Ren [2008]), and a "globalization" step based on spectral clustering is added. These operations are mainly hand-crafted, although there is some optimization of cue combination coefficients.

There have been a number of papers that have considered more wholesale learning approaches, taking as input an image patch and predicting the presence/absence of a boundary at the centre pixel of the patch. For example Dollar et al. [2006] consider a large number of generic features such as gradients and differences between histograms at multiple locations, orientations and scales, and a probabilistic boosting tree is used as a classifier. Prasad et al. [2006] use raw patches aligned to the Canny edge direction, and a SVM classifier. Kokkinos [2010] considers the gPb pipeline, but uses Adaboost to learn the local cues (which are there discriminatively compressed SIFT-descriptors), with an approximate F-score used as the metric to optimize the model. Both Mairal et al. [2008] and Ren and Bo [2012] have used representations based on sparse coding, either directly (the former) or via pooling over oriented half-discs (the latter) to train linear classifiers. The approach in Mairal et al. [2008] considers only image positions detected as contours with the Canny edge-detection method. These positions are in training divided into two categories, those agreeing well with human annotations, and those not, and generic and class-specific sparse coding dictionaries are built of data around the locations. The prediction is then based on the output of a (linear) classifier operating on data reconstruction errors using the dictionaries. The sparse code gradients (SCG) method [Ren and Bo, 2012] (again) considers the gPb detection pipeline, but replaces the hand-crafted gradient measures by representations learned specifically for the task, (this time) by using sparse coding methods. This approach currently has the best performance according to the results at Martin et al. [2013b].

In contrast to these learning approaches, our method builds in the idea of filters with a squaring non-linearity (from the gated MRF work), but also allows several layers of adaptive "deep" nonlinear processing before the classification stage. To ensure each decision takes into account image information from multiple scales and many semantic levels effectively, such layered processing is done via multiple different but interlinked streams, each corresponding to networks with different depths. In order to make the learning and inference effective and fast, we use tiled-convolutional feature sharing strategies in our networks[5].

[5]The hidden units are typically less correlated in such networks compared to those in fully-

Another line of work has addressed linking together edge fragments in order to create extended smooth contours. See e.g. Parent and Zucker [1989] for early work, and Zhu and Shi [2007] for a more recent approach. See the review in Arbelaez et al. [2011] for further references. We also note that region segmentation is closely related to boundary detection, although contour detectors do not necessarily produce closed contours which partition the image into regions.

The boundary labelling problem (classifying each pixel as lying on a boundary or not) can be generalized to the pixel labelling problem (classifying each pixel into one of a given set of labels). For example, the "segmentation" challenge in the PASCAL VOC competition [Everingham et al., 2010] labels each pixel as belonging to one of 20 object classes, or background. Conditional random field (CRF) methods are often used to tackle this problem, due to correlations between nearby labels. One could also consider CRFs for boundary detection, e.g. to allow for multiple high-probability configurations, although note that evaluation method for the BSDS dataset (described in Martin et al. [2004]) only utilizes a per-pixel confidence score. This is a weakness related to the evaluation protocol (see Hou et al. [2013] for an analysis of issues with this protocol), which however is the best available at the moment.

5.3 Experiments

We consider a number of experiments to evaluate our boundary prediction networks, each using the standard datasets and benchmark methodologies, as described in section 5.3.1. The supervised and unsupervised network training procedures are described in section 5.3.2. We have two main goals in our experiments: The first is to evaluate the performance of our methods in comparison to other relevant methods, and second assessing the influence of architectural and other hyperparameter settings contributing to the results. The results for shallow and deep networks are described in section 5.3.3.

5.3.1 Data

We consider the BSDS500 dataset (with main focus on its grey-scale version) and benchmark in our experiments. The dataset consists of 500 natural images and

convolutional ones, which makes faster and thus more effective learning. Typically more parameters are needed, but for many applications, including this one, that is not problematic.

associated boundary annotations by several humans, and is divided into three subsets: training, validation, and test. We follow strictly the protocol in the benchmark, including not using the test set for model development and selection. Examples of test data (grey-scaled, cropped) are shown in Figure 5.21, and also in Appendix D.4.

The predictions of each method are evaluated by the standard BSDS protocols, involving computation of a precision-recall (P-R) curve, as explained in Martin et al. [2004]. As in Arbelaez et al. [2011], the P-R curve can be summarized by computing the F-measure score (the harmonic average of precision and recall) at a particular threshold. This threshold can either be optimized across the data set (ODS), or on a per image basis (OIS). The F-score is considered the main metric of the benchmark (see for example Martin et al. [2013a] and Martin et al. [2013b]). The P-R curve can also be summarized by the average precision (AP).

5.3.2 Training of the models

The mcRBM/mcDBN models were trained using stochastic gradient ascent for approximate maximum likelihood learning based on the FPCD [Tieleman and Hinton, 2009], with implementation inspired by the TmPoT training code [Ranzato et al., 2010b]. The mean and covariance filters were initialized to small random values, the mean and covariance biases were set to -2 and to 2, respectively, and their learning rates assumed the ratios 0.05:0.0025:0.0025:0.0005, respectively, similar to in Ranzato and Hinton [2010]. Only the regular parameter learning rates were annealed, using $\frac{1}{t}$-type annealing, starting at epoch 200, and ending at 0.25 of the initial rates by the end of training at epoch 800. The 32 negative particles (of size 142×142) were drawn and updated by a single step of HMC-sampling with 20 Leapfrog steps, with step-size set automatically to maintain 90 percent sample accept rate according to an exponentially weighted moving average with smoothing factor of 0.9. We used a L_1-weight decay, with rate 0.001. The mcDBNs were trained in the usual greedy manner, layer-by-layer, using exactly the same hyperparameter settings as above, except the all of parameters had the same learning rate, 0.00025 divided by the number of feature plane grid locations (as in the shallow models).

The training of each boundary prediction network is based on a stochastic gradient ascent algorithm, optimizing a function consisting of a sum of two terms:

the supervised log-likelihood $\mathcal{L}$ of the training data, and a regularization term of λ times the L_2-norm of the parameters, with $\lambda = 0.001$. The data log-likelihood $\mathcal{L} = \sum_{n,i} y_i^{(n)} \log u_i^{(n)} + (1 - y_i^{(n)}) \log(1 - u_i^{(n)})$, where $y_i^{(n)}$ denotes the label of the i^{th} contour unit in n^{th} training image, and $u_i^{(n)}$ denotes network prediction for the unit, according to equation (5.4). The read-out weights were initialized to small random values, and the biases to zeroes except for the contour bias g, which was set so that the sigmoid function evaluated at that value was equal to the overall probability of a contour in the training data.

The shallow model parameters had equal learning rates, starting from $\alpha = \frac{0.05 \cdot \beta}{M \cdot T}$, where M denotes the batch size, and T the number of sites the parameter was shared over per image. β was set to 1 and 10 for the feature extraction and read-out parameters, respectively. In the deeper models, the learning rates in the shallow stream were set to 0.5 times those of the above, and those for the deeper stream parameters were set with $\beta = 5$, and setting T as for the shallow layer. Each epoch used as many batches as there were training images. Each batch was comprised of $M = 8$ images of size 142×142-sized randomly cropped from the training images. The learning rates were annealed after 700 epochs using a $\frac{1}{t}$-type annealing, ending at $0.1 \cdot \alpha$ at epoch 800. We used a momentum of 0.4 for the first two epochs, and 0.9 for the later ones.

5.3.3 Results

We discuss the results for shallow and deep architectures in turn. In addition to the different stream architectures, we also assess (i) the relative importance of the mean and covariance units, and (ii) the effect of generative pre-training [Hinton et al., 2006]. In the latter case the three options were (a) initialization of the parameters to those of the mcRBM/mcDBN followed by supervised fine-tuning, (b) fixing (i.e. freezing) the learned unsupervised mcRBM/mcDBN parameters, and (c) starting them from random settings (initialized as those in the generative training). We did not explore the full combinatorial space of settings, but focused on the most important cases.

5.3.3.1 Dissecting Boundary Prediction with Shallow Networks

We considered first boundary prediction with the shallow networks. Table 5.1 summarizes the results, and Figure 5.12(top) shows precision-recall curves, for

the BSDS500 test set. We observe that when the parameters are not fine-tuned [mcRBM fixed entries], the covariance units tend to carry more contour information, as the performance without the mean units nearly matches that of the full model, while performance without the covariance units is clearly weaker.

When fine-tuning is in place, the performances increase significantly, especially so for the mean-only model, but the relative order of the models is still maintained. Interestingly, initialization of the feature extraction parameters from the generatively trained mcRBM yields better results than random initialization.

Figures 5.13 and 5.14 visualize the full set of weights in the full shallow model, without and with fine-tuning of the feature extraction (mcRBM) parameters. The fine-tuning results in clear changes in the filter appearances and properties. We can observe in Figure 5.14 for example that the mean feature extraction filters change in appearance to be more localized, and the associated read-out weights become larger in amplitude, thus influencing the prediction more. In Figure 5.13, the checker-patterned fine-detail covariance feature extraction filters have mostly disappeared after fine-tuning. Figures 5.15, 5.16, 5.17,and 5.18 compare hidden unit activations as a response to different image patches (as in Figures 5.3 and 5.4) under the two different models. From the figures we can see sparser hidden unit activations by the model with the fine-tuned parameters in general. Figure 5.19 visualizes contour prediction by the model with the fine-tuned feature extraction parameters for several positions, and also the contributions by the covariance, and the mean hidden unit features to the prediction. There is a surface boundary only under Position 1, and the model is producing strongest predictions there. Although both the covariance and the mean contributions are markedly high there, the former is clearly more distinct. Figure 5.20 visualizes results by the model without fine-tuned feature extraction parameters. The predictions under Position 1 are now clearly weaker, and much higher boundary predictions are obtained under the other positions, which do not actually contain a boundary. Also the mean features are relatively meaningless, with very similar predictions for all of the sites in any position. The largest activation, and the variation in it occurs for Position 3, which does not even contain a boundary. These results further highlight the importance of the fine-tuning, and that the mean features, as expected, are not having a large role in the generation of boundary sites.

Figure 5.21 shows example inferences for a large test image. For comparison, Canny edge detection (obtained with default settings in the matlab implemen-

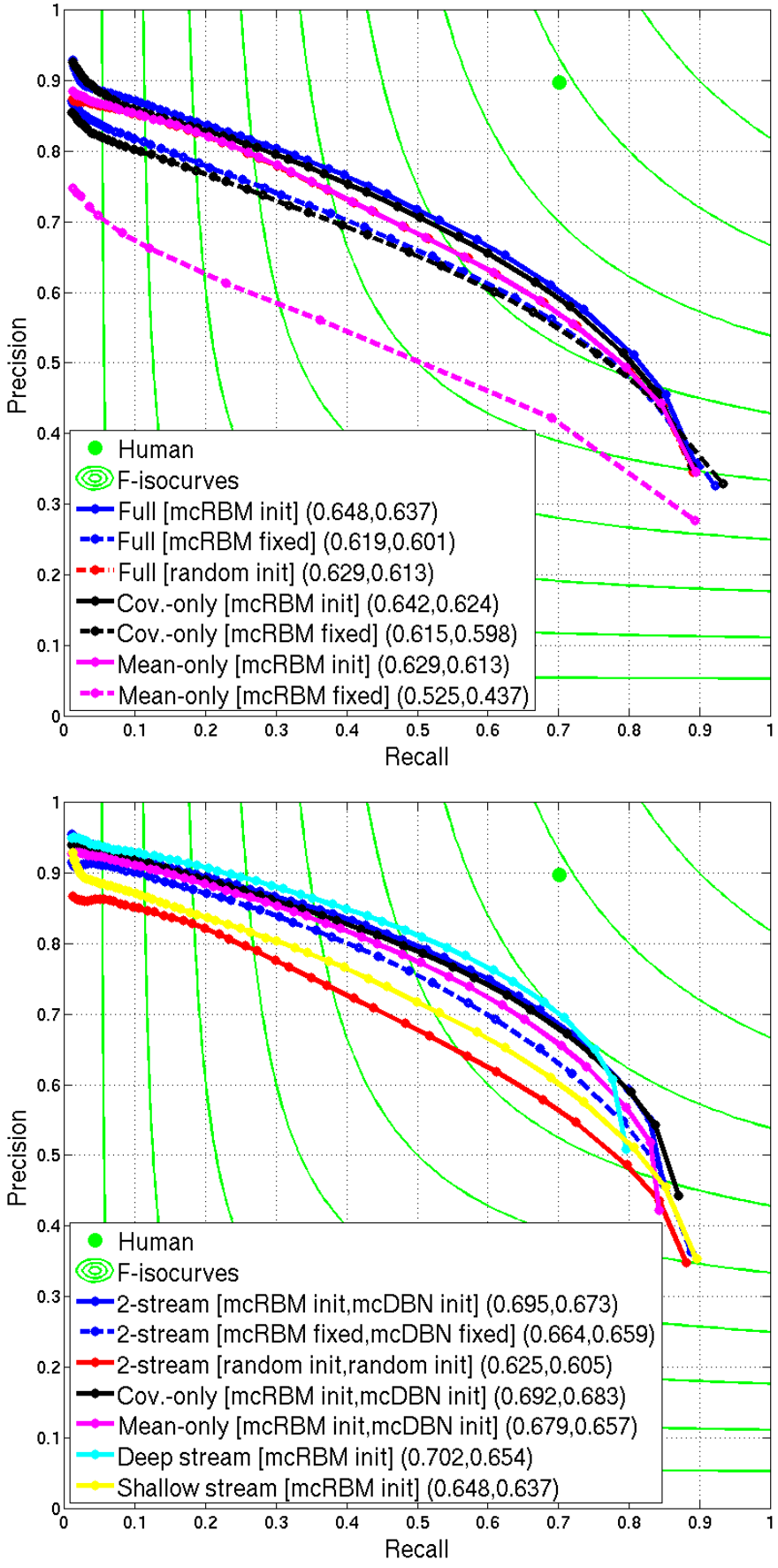


Figure 5.12: Contour prediction results on the BSDS500 as measured by precision-recall curves, with shallow models (top), and with deeper models (bottom). The numbers in the legends denote the maxima of the curves w.r.t. the ODS F-measure (left), and the average precision (right).

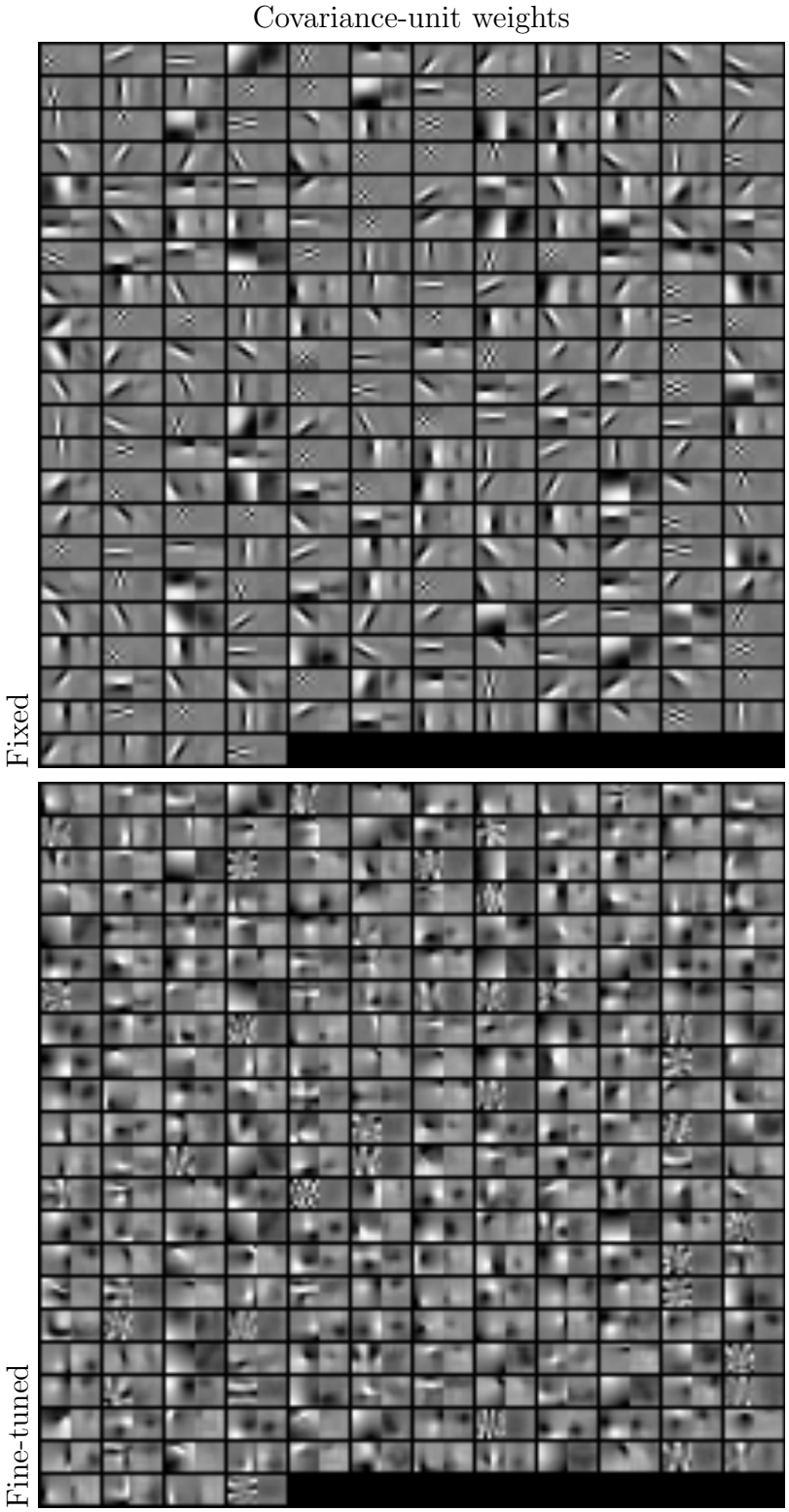


Figure 5.13: All of the shallow model covariance unit weights, with and without mcRBM-parameter fine-tuning. The small black-boxes each contain the image feature extraction (left part) and contour prediction (right part) filter pairs. The filters connecting to the image data are normalized invidually, whereas the filters connecting to the contour data are globally normalized, so as to be on the same scale and to fill the full intensity range.

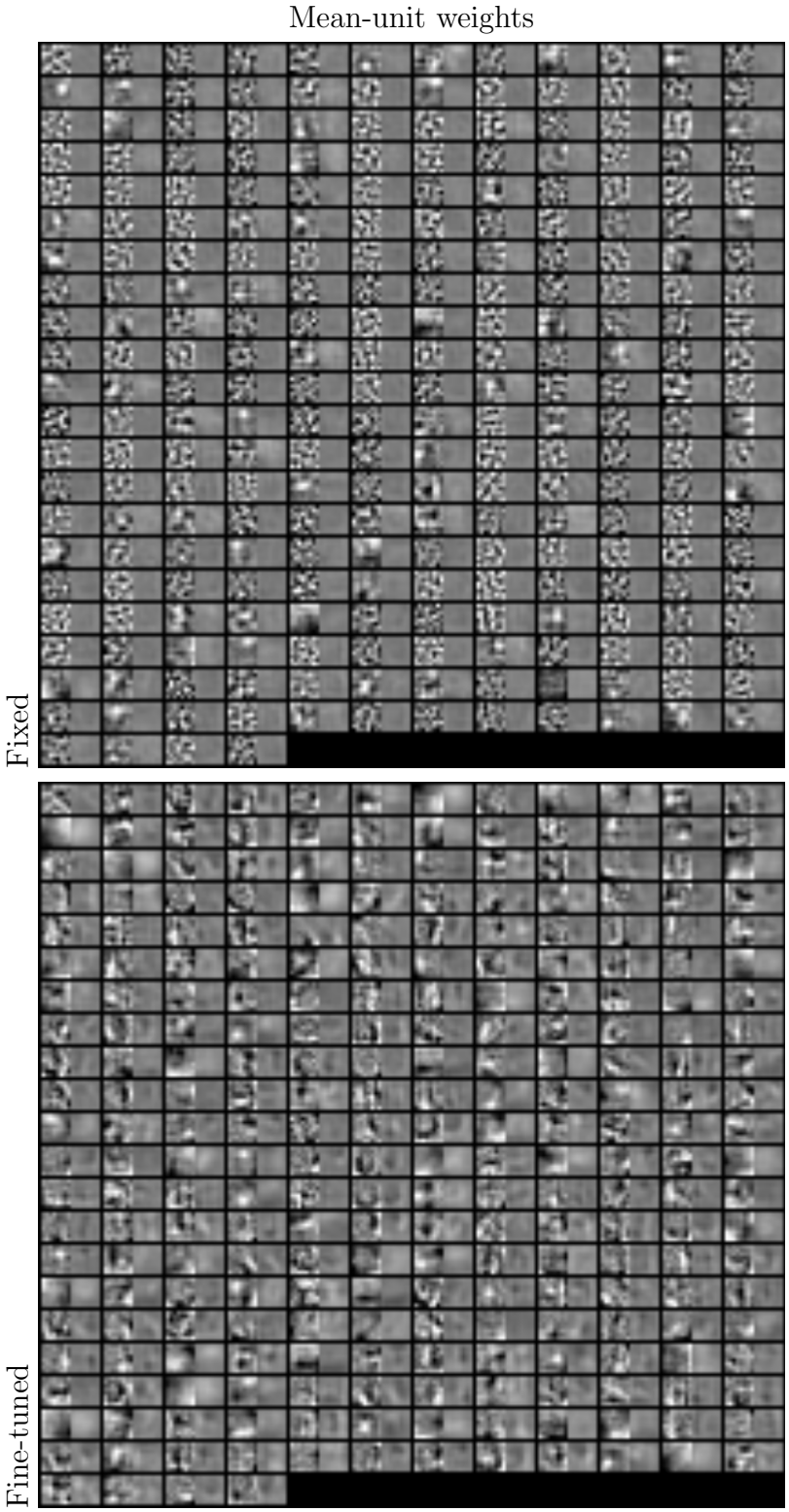


Figure 5.14: All of the shallow model mean unit weights, with and without mcRBM-parameter fine-tuning. The small black-boxes each contain the image feature extraction (left part) and contour prediction (right part) filter pairs. The filters connecting to the image data are normalized invidually, whereas the filters connecting to the contour data are globally normalized, so as to be on the same scale and to fill the full intensity range.

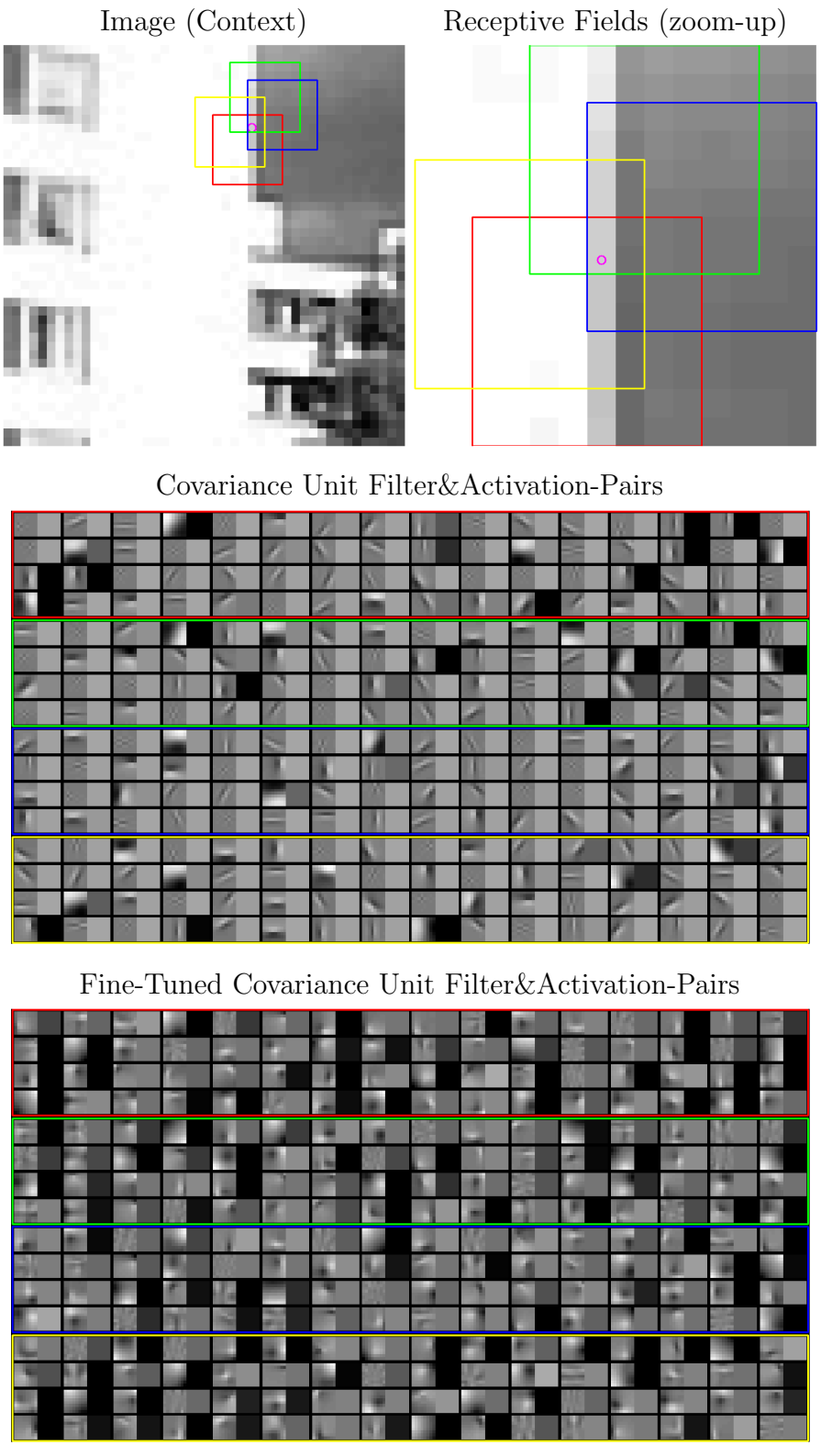


Figure 5.15: Covariance hidden unit states under an mcRBM trained generatively vs. discriminatively for contour prediction. The rectangles in the input image denote boundaries of receptive fields of example hidden unit stacks, under each of the four different parameter sets. Together the stacks form all of the hidden units which connect to the visible unit position marked with a magenta-coloured circle. The weights of these sets, and the activations of the units at the positions are visualized in horizontal pairs, with a pair per black rectangle. Best viewed on screen.

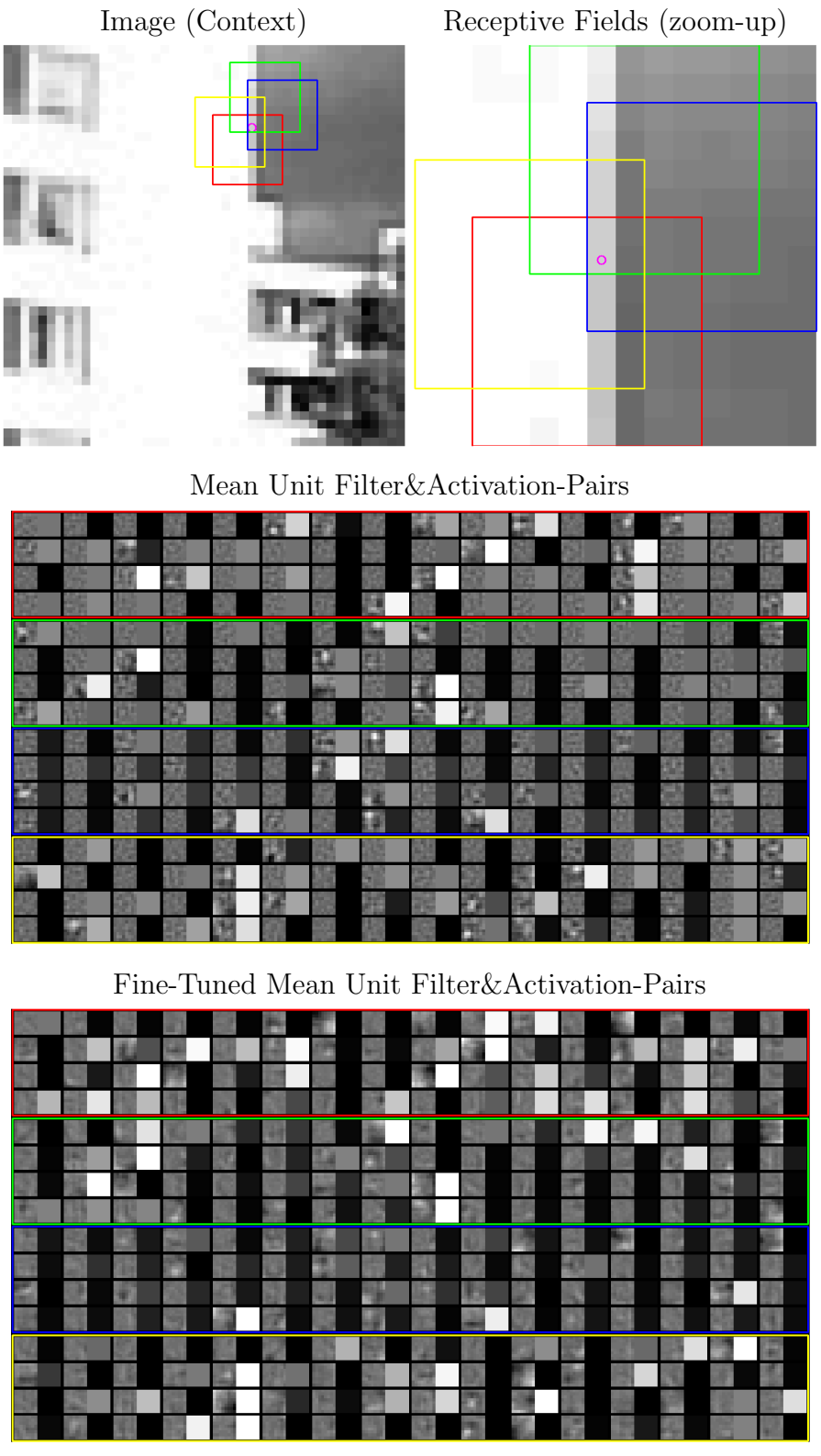


Figure 5.16: Mean hidden unit states under an mcRBM trained generatively vs. discriminatively for contour prediction. The rectangles in the input image denote boundaries of receptive fields of example hidden unit stacks, under each of the four different parameter sets. Together the stacks form all of the hidden units which connect to the visible unit position marked with a magenta-coloured circle. The weights of these sets, and the activations of the units at the positions are visualized in horizontal pairs, with a pair per black rectangle. Best viewed on screen.

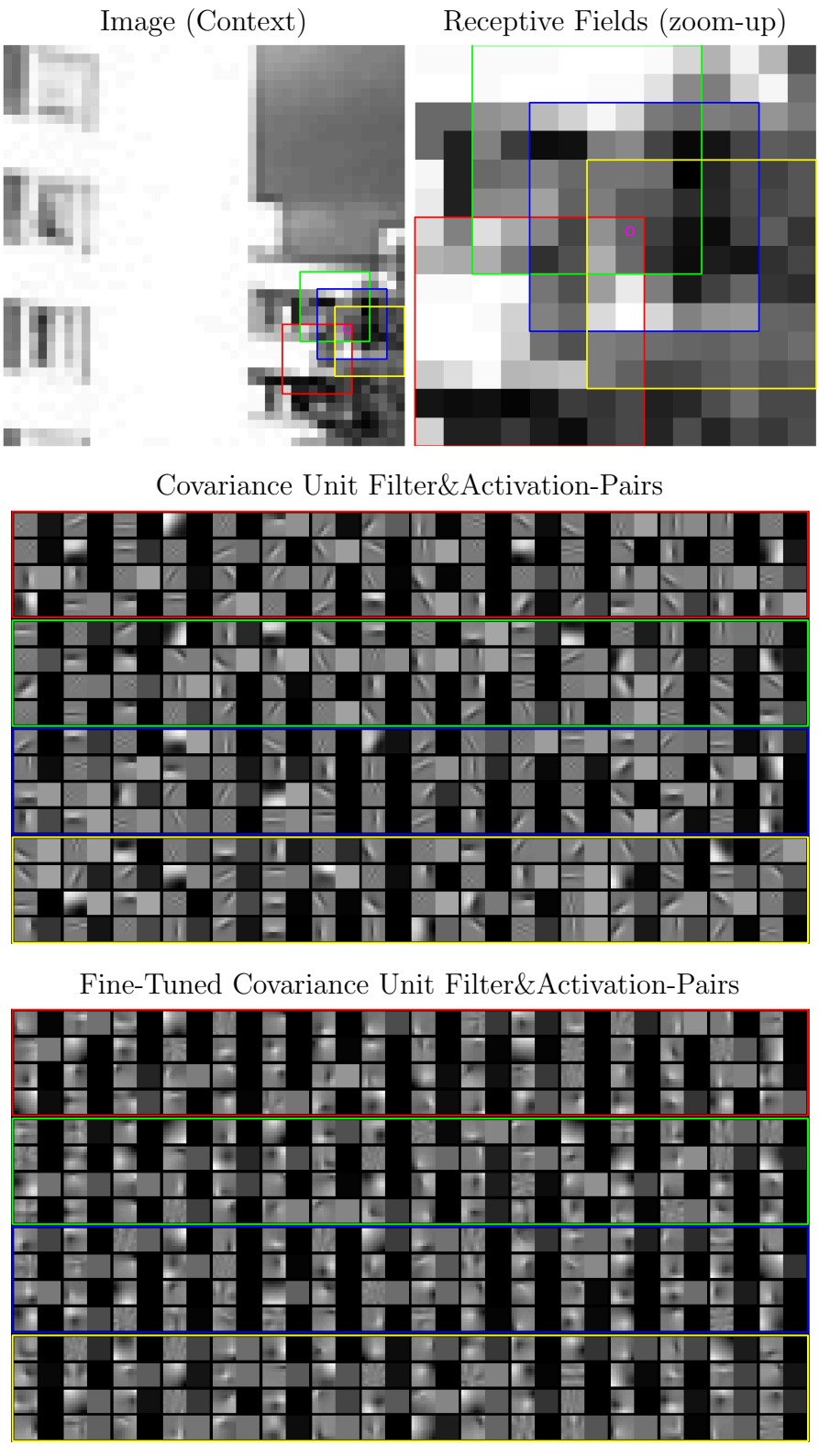


Figure 5.17: Covariance hidden unit states under an mcRBM trained generatively vs. discriminatively for contour prediction. The rectangles in the input image denote boundaries of receptive fields of example hidden unit stacks, under each of the four different parameter sets. Together the stacks form all of the hidden units which connect to the visible unit position marked with a magenta-coloured circle. The weights of these sets, and the activations of the units at the positions are visualized in horizontal pairs, with a pair per black rectangle. Best viewed on screen.

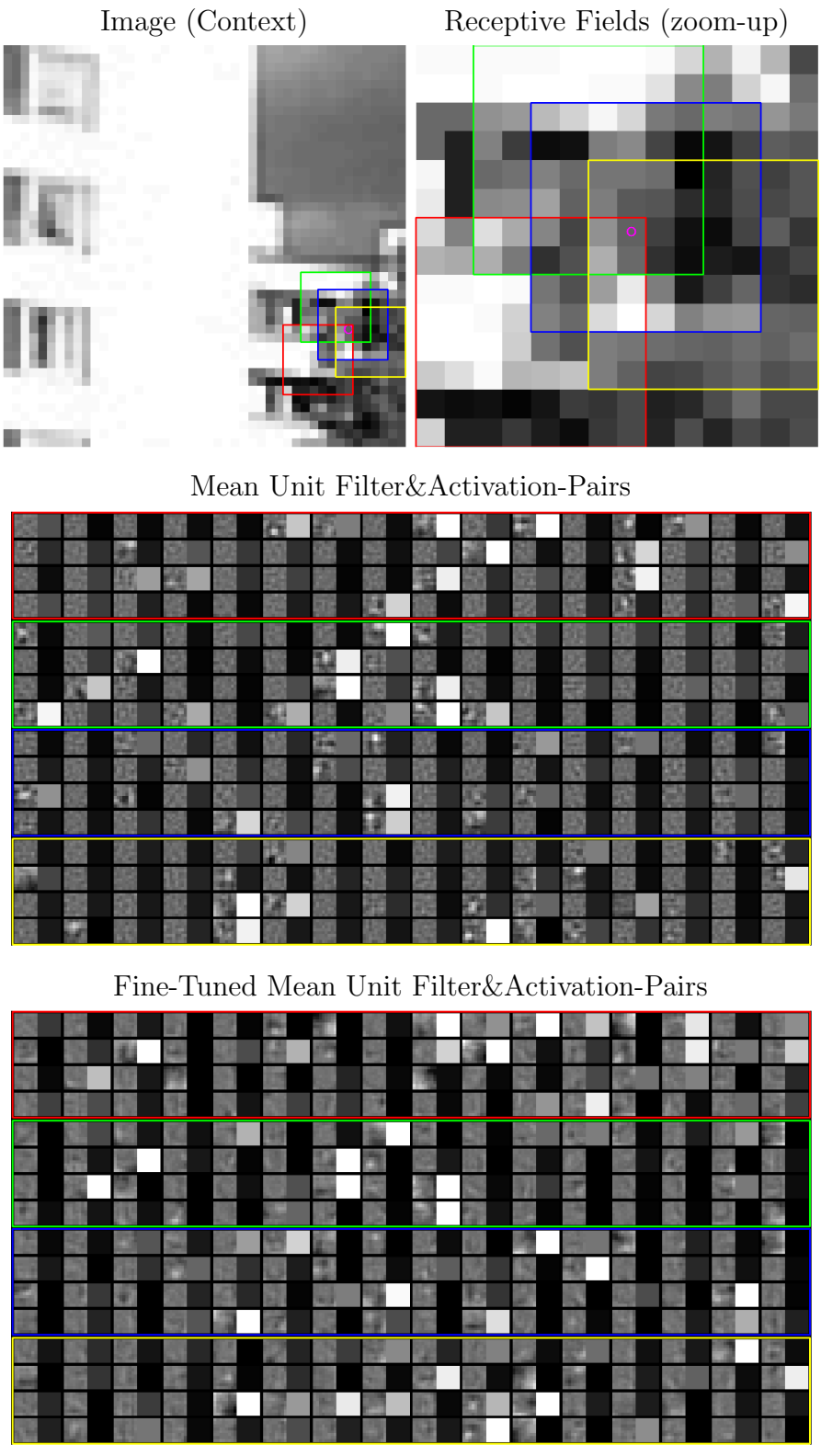


Figure 5.18: Mean hidden unit states under an mcRBM trained generatively vs. discriminatively for contour prediction. The rectangles in the input image denote boundaries of receptive fields of example hidden unit stacks, under each of the four different parameter sets. Together the stacks form all of the hidden units which connect to the visible unit position marked with a magenta-coloured circle. The weights of these sets, and the activations of the units at the positions are visualized in horizontal pairs, with a pair per black rectangle. Best viewed on screen.

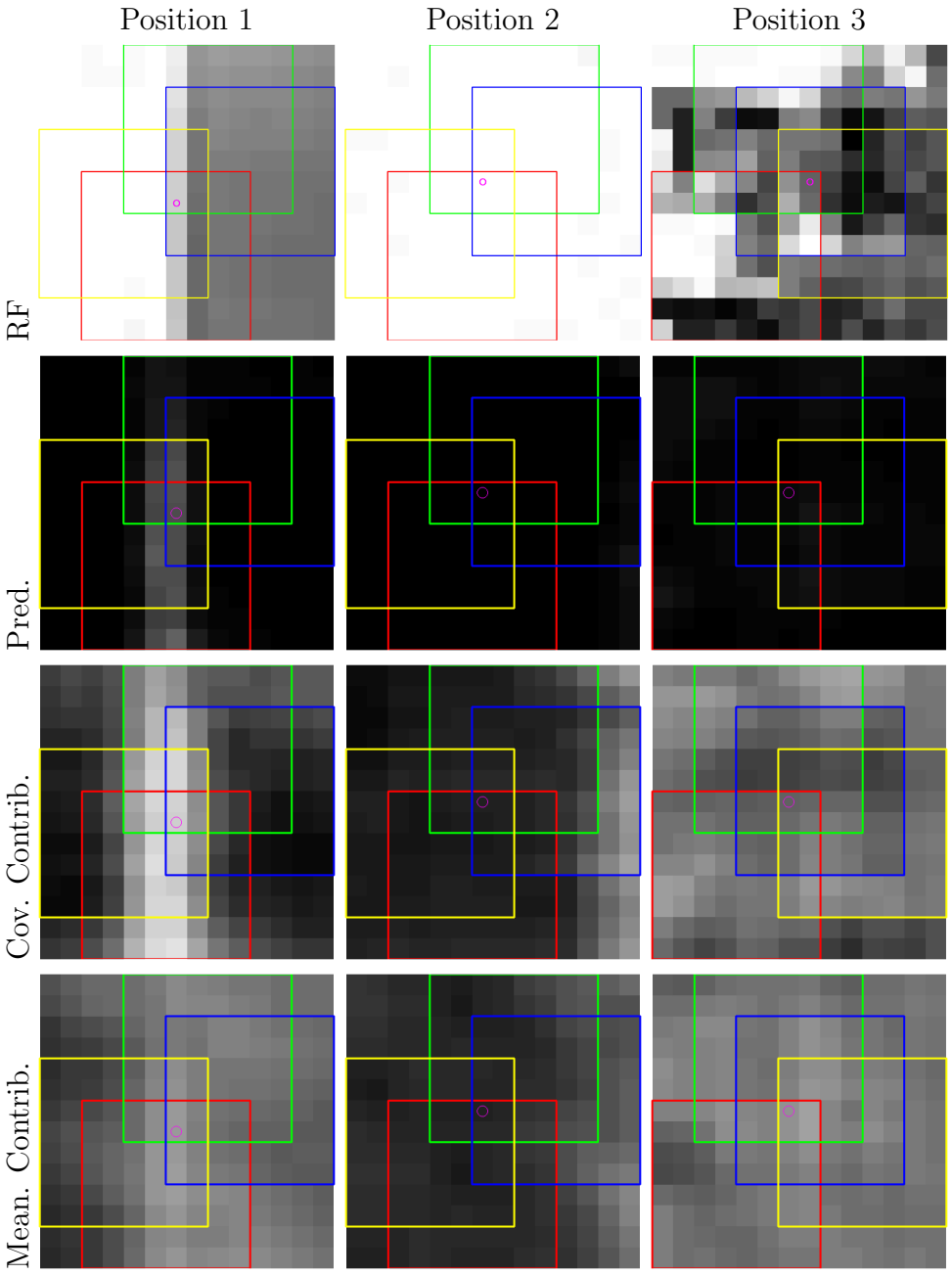


Figure 5.19: Contour prediction properties under a shallow full model [mcRBM init]. The prediction value is visualized with intensity, with white denoting probability of one for boundary, and black denoting probability of zero for boundary (same scale for the different positions). The feature contributions to site input are also visualized with intensity, whiter meaning greater input (same scale for the different positions). The rectangles denote boundaries of receptive fields of example hidden unit stacks, under each of the four different parameter sets. Together the stacks form all of the hidden units which connect to the visible unit position marked with a magenta-coloured circle. See the Figures above for the corresponding hidden unit filters and activations. Best viewed on screen.

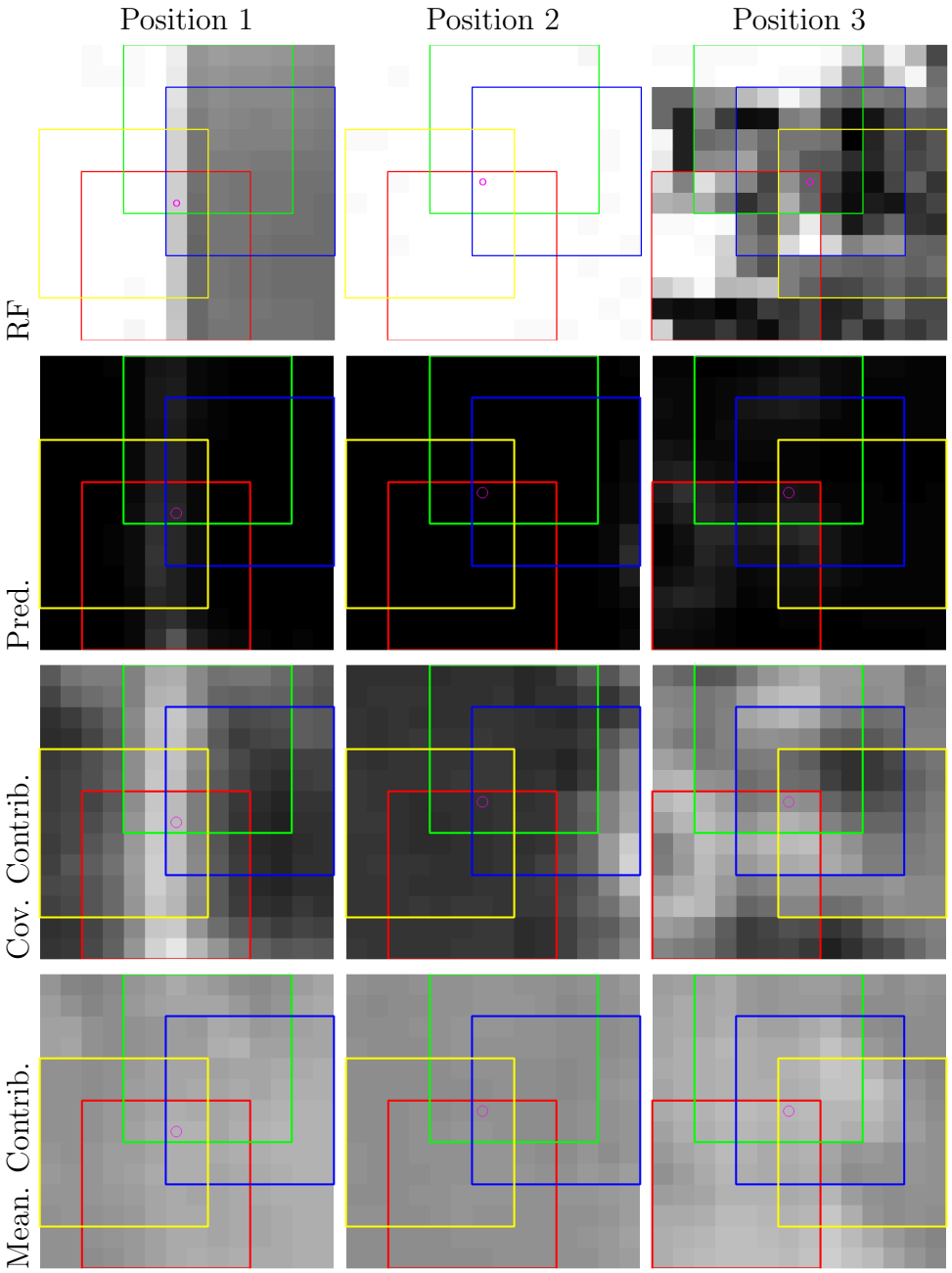


Figure 5.20: Contour prediction properties under a shallow full model [mcRBM fixed]. The prediction value is visualized with intensity, with white denoting probability of one for boundary, and black denoting probability of zero for boundary (same scale for the different positions). The feature contributions to site input are also visualized with intensity, whiter meaning greater input (same scale for the different positions). The rectangles denote boundaries of receptive fields of example hidden unit stacks, under each of the four different parameter sets. Together the stacks form all of the hidden units which connect to the visible unit position marked with a magenta-coloured circle. See the Figures above for the corresponding hidden unit filters and activations. Best viewed on screen.

	F-score		
Model type [feature extraction setting]	**ODS** (P,R)	OIS (P,R)	AP
Full [mcRBM fixed]	0.619 (0.690,0.562)	0.643 (0.739,0.569)	0.601
Full [random init]	0.629 (0.689,0.579)	0.651 (0.737,0.583)	0.613
Full [mcRBM init]	0.648 (0.699,0.604)	0.668 (0.745,0.605)	0.637
Covariance-only [mcRBM fixed]	0.615 (0.691,0.555)	0.641 (0.739,0.565)	0.598
Covariance-only [mcRBM init]	0.642 (0.707,0.587)	0.660 (0.740,0.596)	0.624
Mean-only [mcRBM fixed]	0.525 (0.667,0.432)	0.539 (0.670,0.451)	0.437
Mean-only [mcRBM init]	0.629 (0.689,0.578)	0.652 (0.736,0.585)	0.613

Table 5.1: Statistics on boundary prediction under the **BSDS500** test set by the **shallow** prediction model by using different feature sets.

tation) which produces a binary edgemap output is also shown. The feature extraction parameters of the shallow network is trained with the mcRBM parameter initialization, and fine-tuning. We see that it correctly places probability mass on locations where humans have placed annotations, but also in regions with repeated local structure. See Appendix D.4 for more examples.

5.3.3.2 Dissecting Boundary Prediction with Deeper Networks

Table 5.2 summarizes the test-set results for the deep stream and two-stream networks. Our models were trained on the 200 training images, and hyperparameter selection (e.g. setting learning rates, etc.) was done on the validation set. Similar to the earlier experiments, we can see under any particular initialization setting, that the covariance-only models perform better than the mean-only models. We can also see that the generative pre-training helps the full two-stream model in an even more pronounced way than the shallow-stream model (compare random init. vs. mcRBM and mcDBN init). Allowing fine-tuning of the feature extraction parameters (compare fixed vs. init for a particular model setting) again results in improved performance. See also Figure 5.12 (bottom) for P-R-curves of many of these cases for further verification.

When comparing the performance summaries related to the different streams, it is noticeable that the deep model is clearly better than the shallow model. This can be also seen in the Figure 5.21 inference example, with improved ability to filter out predictions in textured regions containing many edges. We can see for

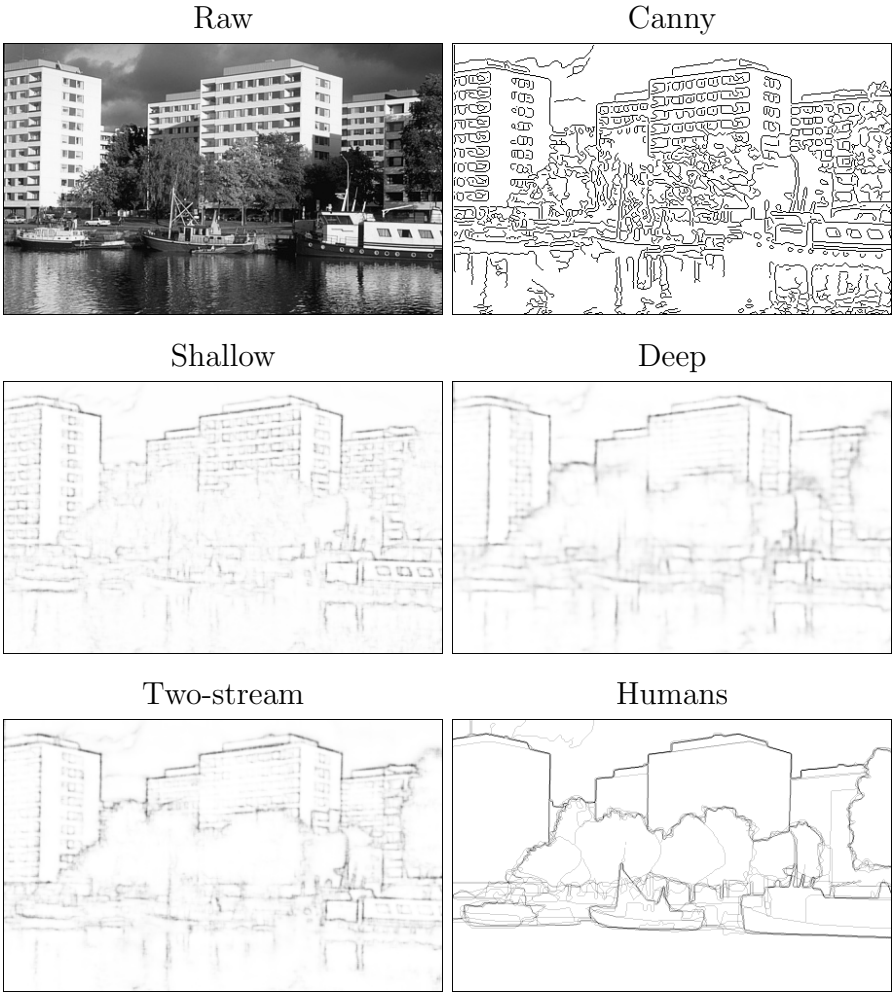


Figure 5.21: Contour prediction result example on the grey-scale BSDS500. The prediction value is visualized with intensity, with black denoting probability of one for boundary, and white denoting probability of zero for boundary. Best viewed on screen.

example many building windows appearing in a locally repeated fashion being removed. This can also be seen in the deep-stream contributions in Figure 5.22. See Appendix D.4 for more inference examples, and D.5 for dissections of the two-stream model inference with respect to different components.

Although the numerical performance of the deep model is comparable to the two-stream model with respect to the ODS and OIS metrics (across the different cases in Table 5.2), the average precision is clearly worse. We can see from the Figure 5.12 (bottom) that the two-stream model is able to obtain higher recall rates, with the P-R curve of the deep-only model dropping off earlier. Our best-performing model (ODS 0.702, OIS 0.718, AP 0.687) using the binary annotation data was obtained by taking the model which performed best on the validation set (two-stream with mcRBN and mcDBN init), and training it on all 300 train+val images. Note that this is following the guidelines precisely, using the train/val splits for hyperparameter selection, and then producing a final model using train+val. This model outperforms gPb [Arbelaez et al., 2011] on all three measures, and SCG (local) on ODS and OIS.

Both gPb and SCG (global) use a computationally expensive "globalization" step which involves computing image-sized eigenvectors, and a rather ad hoc integration of this information with the local predictions tuned to directly optimize the F-scores. Our unoptimized implementation of the two-stream model inference takes 0.1 to 0.2 s per test image. This compares very favorably with the figures quoted in Lim et al. [2013], which are 60s (gPb local), 100s (SCG local), 240s (gPb global), 280s (SCG global) and 1 sec (Sketch tokens [Lim et al., 2013]). We have confirmed the gPb global figure on our local machines. Our implementation is on a GPU, but this emphasizes the easily parallelizable nature of our method. We note that Catanzaro et al. [2009] have reduced the runtime of gPb global to 1.8s using GPUs, so we are still a factor of 10 faster than them. In general the evidence is that one may typically obtain 10x speedups by going to GPUs, when comparing tuned implementations for *both* CPU and GPU [Lee et al., 2010].

5.3.3.3 Tuning the Prediction Performance

Here we describe four simple modifications to the learning described above in order to tune the networks for optimal prediction performance. As before the model selections were done using the 'val' set, and the models trained on 'train+val' use the hyperparameter settings of the respective model trained on 'train' and se-

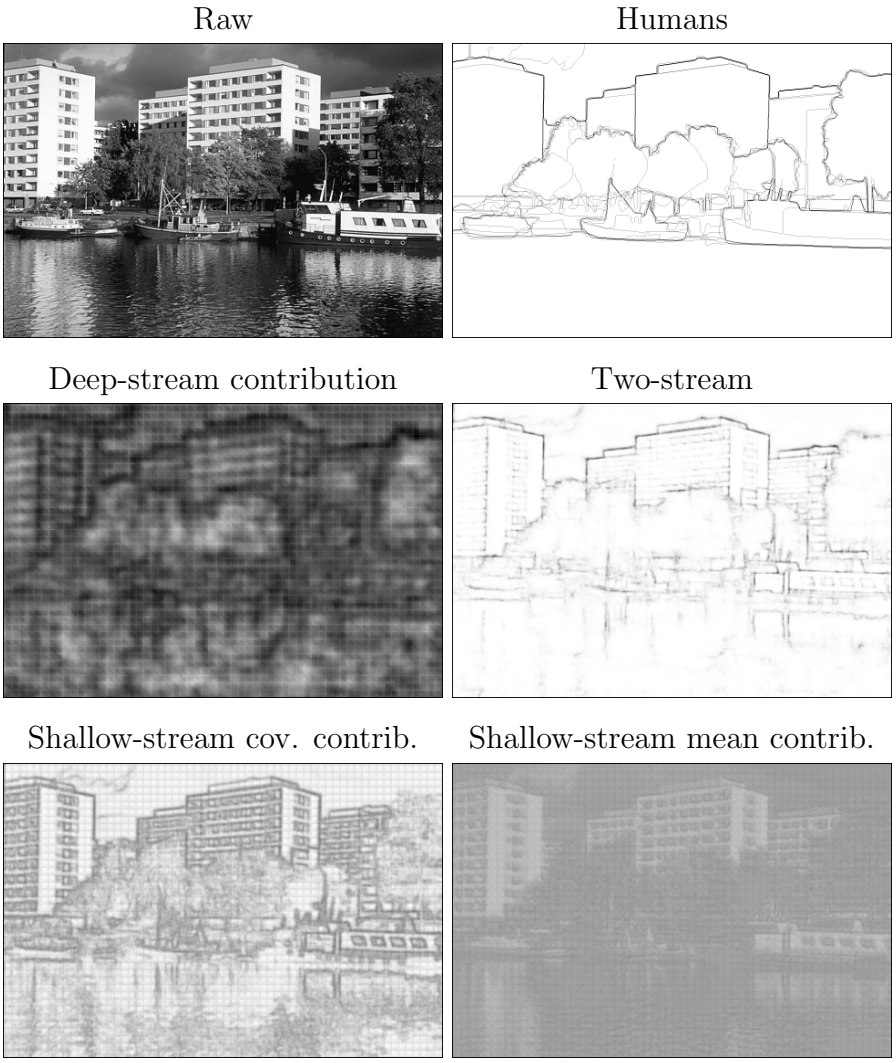


Figure 5.22: Dissecting a contour prediction result on the grey-scale BSDS500. The prediction value is visualized with intensity, with black denoting probability of one for boundary, and white denoting probability of zero for boundary. The stream contributions are normalized to be on the same scale, which also fills the full intensity range. Best viewed on screen.

Model type [shallow params,deep params]	F-score **ODS** (P,R)	OIS (P,R)	AP
Two-stream [mcRBM fixed,mcDBN fixed]	0.664 (0.711,0.623)	0.682 (0.731,0.640)	0.659
Two-stream [random init,random init]	0.625 (0.689,0.572)	0.649 (0.733,0.583)	0.605
Two-stream [mcRBM init,random init]	0.667 (0.703,0.634)	0.684 (0.733,0.642)	0.663
Two-stream [mcRBM init,mcDBN init]	0.695 (0.721,0.670)	0.709 (0.747,0.674)	0.673
The above with 300 training images	**0.702** (0.731,0.676)	**0.718** (0.762,0.678)	**0.687**
Deep stream [mcRBM fixed,mcDBN fixed]	0.625 (0.630,0.619)	0.641 (0.638,0.643)	0.537
Deep stream [mcRBM init,mcDBN init]	0.702 (0.708,0.696)	0.715 (0.734,0.697)	0.654
Cov.-only two-stream [mcRBM fixed,mcDBN init]	0.686 (0.719,0.656)	0.703 (0.752,0.660)	0.667
Cov.-only two-stream [mcRBM init,mcDBN init]	0.692 (0.734,0.655)	0.708 (0.757,0.666)	0.683
Cov.-only deep stream [mcRBM init,mcDBN init]	0.696 (0.711,0.681)	0.710 (0.739,0.683)	0.652
Mean-only two-stream [mcRBM fixed,mcDBN init]	0.646 (0.672,0.622)	0.663 (0.705,0.625)	0.603
Mean-only two-stream [mcRBM init,mcDBN init]	0.679 (0.715,0.646)	0.696 (0.748,0.651)	0.657
Mean-only deep stream [mcRBM init,mcDBN init]	0.679 (0.707,0.653)	0.694 (0.723,0.667)	0.627
Canny [Arbelaez et al., 2011, Tab. 1]	0.60	0.63	0.58
gPb [Arbelaez et al., 2011](from Ren and Bo [2012])	0.69	0.71	0.67
SCG (local) [Ren and Bo, 2012]	**0.69**	**0.71**	**0.71**
SCG (global) [Ren and Bo, 2012]	**0.71**	**0.73**	**0.74**
Human [Arbelaez et al., 2011]	0.80	0.80	

Table 5.2: Statistics on boundary prediction for the **BSDS500** (grey-scale) test set by the **deeper** prediction models by using different feature sets and architectures.

lected on the 'val'. Two of the modifications involve changes in setting the batch data for training. When training the networks above, the images in each batch were set by cropping patches of a certain size (142×142) from any of the training images from random locations, and the labels for each of the patches were set by randomly choosing a single annotation from all of the annotations. The first of the modifications sets the labels as averages of the contour annotations. This reduces noise in the learning, and our results were consistently better with the approach[6]. See Appendix D.2 for further information and discussion on using such data in training and also in testing.

The second modification standardizes the image patches, subtracting their mean from them, and dividing by their standard deviation. The motivation for this standardization is to make the network more robust to data variations (shifts and scaling in global intensity within the extracted training patches) not considered to be relevant to the problem of deciding whether a site should be a boundary or not.

The third modification involves encouraging hidden unit activation levels (measured as an exponentially weighted moving average of the observed levels with smoothing factor of 0.9) to be at/near specific levels, by adding to the objective function cross-entropy penalties between the target and measured levels, as in (2.51). This was applied to the final hidden unit layer in the deep stream with target activation level of 0.1, with non-sparsity penalty of 0.001, and also to the mean hidden units (with target activation level of 0.1, with non-sparsity penalty of 0.01).

The fourth modification (which we call shift-averaging) addresses the lack of full translation equivariance. The developed technique is related to the cycle-spinning method by Coifman and Donoho [1995] who share a similar motivation but for (non-steerable) wavelet-based image denoising. They address the issue by applying wavelet-domain denoising for all circlularly shifted versions of a noisy image, unshift and average the results (as the denoised image). We similarly 'average out' shift-dependence by averaging unshifted prediction results on shifted versions of the test image at hand. Note that only very few shifts out of all possible shifts typically need to be considered for obtaining all possible network prediction results (due to the diag-tiled-convolutional sharing); this is, loosely speaking, upper-bounded by the number of pixels within input image region a

[6]The learning rates were also scaled by a constant.

contour unit is affected by. Also instead of circular shifting we pad (further) the test images by mirror reflection according to the shift amounts (and crop them out after prediction).The modification makes the full prediction slower but fortunately the computations for the different shifts-specific images can be done in a parallel fashion[7]. Fully-convolutional model would have the translation-equivariance property built-in. However, training natural image models with such parameter sharing has been reported to be harder [Ranzato et al., 2010b] and generative pre-training was shown to be important for our networks. Similar averaging with respect to orientation and scale might provide further improvements, but such experiments, along with fully-convolutional and other transformation equivariant versions of our networks, are left for future work.

As can be seen from Table 5.3 where the results on the BSDS500 dataset are summarized, by using only the first three modifications, our approach yields the same or slightly higher performance in terms of the F-score (ODS) than SCG (global), and is thus the highest ranking method reported on the benchmark with also fastest reported inference time (which is the same as reported above). By considering also the fourth modification (where we consider all shifts within $[0, 1, \ldots, 13]^2$) we clearly outperform the state-of-the-art in terms of the ranking metric F-score (ODS).

Table 5.4 compares our prediction results on the BSDS300 data set to the main competitors numerically. For these data, the result images by the competitors were available online (at Martin et al. [2013b]), and we can report their scores to 3 decimal places. Our F-scores (ODS) of 0.696 (basic) and 0.704 (with shift-averaging) obtained on the BSDS300 (grey-scale) test set are the highest reported, with the closest competitors including the gPb-ucm and SCG (global) obtaining the scores of 0.684 and 0.682, respectively [Martin et al., 2013b]. However, as we have done model selection using the BSDS 300 test set (which is the BSDS 500 validation set), our results on it could be optimistic.

We have not redone the dissection computations using these 'tricks'. Reasons for this include that they are expected to be mostly orthogonal to the dissection, and would complicate the procedure by requiring several additional hyperparameters to be chosen, and training modifications to be done.

[7] which implementation is left for future work.

Learning/prediction setting	F-score		AP
	ODS (P,R)	OIS (P,R)	
Basic two-stream (200)	0.695 (0.721,0.670)	0.709 (0.747,0.674)	0.673
Basic two-stream (300)	0.702 (0.731,0.676)	0.718 (0.762,0.678)	0.687
Training on contour-strengths (200)	0.705 (0.727,0.684)	0.719 (0.749,0.690)	0.679
Training on contour-strengths (300)	0.712 (0.731,0.695)	0.726 (0.752,0.701)	0.690
Also with image patch standardization (200)	0.711 (0.735,0.688)	0.725 (0.752,0.701)	0.677
Also with image patch standardization (300)	0.716 (0.745,0.689)	0.729 (0.761,0.700)	0.692
Also with sparsity encouragement (200)	0.711 (0.741,0.684)	0.727 (0.748,0.707)	0.679
Also with sparsity encouragement (300)	**0.719** (0.742,0.697)	**0.733** (0.758,0.710)	**0.697**
Also with shift-averaging (200)	0.721 (0.741,0.702)	0.735 (0.753,0.718)	0.673
Also with shift-averaging (300)	**0.727** (0.745,0.710)	**0.743** (0.761,0.725)	**0.690**
(Literature results where rounding had been done by 2 decimal places):			
Canny [Arbelaez et al., 2011, Tab. 1]	0.60	0.63	0.58
gPb [Arbelaez et al., 2011](from Ren and Bo [2012])	0.69	0.71	0.67
SCG (local) [Ren and Bo, 2012]	**0.69**	**0.71**	**0.71**
SCG (global) [Ren and Bo, 2012]	**0.71**	**0.73**	**0.74**
Human [Arbelaez et al., 2011]	0.80	0.80	

Table 5.3: Statistics on boundary prediction for the **BSDS500** (grey-scale) test set by the **two-stream** prediction models by using different training settings.

	F-score		
Approach on greyscale BSDS300	**ODS** (P,R)	OIS (P,R)	AP
Basic two-stream (200)	0.679 (0.708,0.652)	0.686 (0.731,0.646)	0.657
Training on contour-strengths (200)	0.686 (0.705,0.667)	0.692 (0.730,0.658)	0.658
Also with image patch standardization (200)	0.691 (0.710,0.673)	0.698 (0.736,0.664)	0.659
Also with sparsity encouragement (200)	**0.696** (0.724,0.670)	**0.702** (0.736,0.671)	**0.664**
Also with shift-averaging (200)	**0.704** (0.720,0.689)	**0.714** (0.692,0.714)	**0.653**
gPb-ucm [Arbelaez et al., 2011]	**0.684** (0.710,0.659)	**0.722** (0.740,0.704)	**0.634**
SCG [Ren and Bo, 2012]	0.682 (0.713,0.653)	0.703 (0.747,0.663)	0.723
Human	0.79	0.74	0.73

Table 5.4: Statistics on boundary prediction under the **BSDS300** (grey-scale) test set by various models. The result images by the competition (from which the scores were computed) were obtained from the BSDS300 dataset site [Martin et al., 2013b]. Both the scores of our best networks, and the highest ranking method reported on the benchmark are highlighted.

5.4 Discussion

We have investigated several feed-forward network architectures for visual boundary prediction, each built on top of a diagonally-tiled convolutional mcRBM. The architectures allow end-to-end optimization, and fast and scalable inference for prediction. Our best results are comparable or better to those of the best-performing methods on the grey-scale BSDS500 dataset, and do not require an expensive "globalization" step. Our method is fast, and has a very different architecture from the others proposed in the literature (e.g. both Pb and SCG use hand-crafted oriented half-disc pooling, while ours has no such procedure).

In all of the architectures considered covariance features were shown to be more important than mean features for the task. Deep networks, which were able to discount interiors of textured regions as being boundaries, were clearly better than the more local shallow ones. Combining the shallow and deep streams resulted in additional improvements. We also observed significant benefits of generative pre-training, with marked performance improvements over random parameter initializations. Fine-tuning of the parameters allowed further improvements, and was crucial for the mean-only shallow model, sheding light on what the different types of mcRBM units are doing in the generative setting.

We are currently exploring the extension of the networks to colour data (see Appendix D.3 for preliminary results), and especially to further improved initialization techniques. In this chapter the read-out parameters were initialized to random values. Although greedy modality-specific generative training is possible, a more principled alternative would be to consider a joint model of the image and contour data. In the shallow domain, such a model can be obtained by connecting the (now stochastic) contour units $\mathbf{u}$ to the hidden units of the mcRBM, and defining the energy of the image and contour RBM (icRBM) model as:

$$E_{\text{icRBM}}(\mathbf{h}, \mathbf{v}, \mathbf{u}) = E_{\text{mcRBM}}(\mathbf{h}, \mathbf{v}) - \sum_j h_j^c \mathbf{W}_{\cdot j}^{c\top} \mathbf{u} - \sum_\ell h_\ell^m \mathbf{W}_{\cdot \ell}^{m\top} \mathbf{u} - g \sum_i u_i, \quad (5.5)$$

where $\mathbf{W}_{\cdot j}^c$ and $\mathbf{W}_{\cdot \ell}^m$ denote a filter from the j^{th} covariance, and ℓ^{th} mean hidden unit to contour units, respectively. The same technique can be used to merge information from the different modalities in the deeper layers, both defining instances of dual-wing harmoniums [Xing et al., 2005]. Such joint models would allow for also other kinds of interesting applications, including image-prediction from boundary data (de-sketching) and image completion. For the latter task,

boundary data might be available, and improve performance.

Other potential, and possibly simpler extensions include more-flexible stream integration models, and using transformation-equivariant covariance filters to provide direction-encoding. The networks can also be extended for other image analysis tasks, such as image restoration.

A benefit of the deterministic units in the network is the simplicity associated with learning and inference. However, regardless of how many hidden units are used, and their connectivity structure before the final contour unit layer, decisions at the final layer are always done at each site independently of the other sites. Conceptually this is a problem, for which a solution is to consider stochastic units in place of deterministic units, possibly only in a few layers and feature planes. However, there would need to be stochastic units for which Markov blanket would include other spatially-offset stochastic units for communication of the states, and thus enable 'globalization' (obtaining spatial coherence, here via node state dependence on neighboring/surrounding node states) to happen. The connections between the stochastic nodes could be made undirected in the case of which one would obtain a CRF-structure (see e.g. Sutton and McAllum [2012] for details on CRFs). For a similar performance level, it might be the case that the number of parameters under a deterministic network might become so large that combatting overfitting might become problematic. With the increased representational capacity would come increased computational complexity in learning and inference. However, this is a conceptually simple, yet a statistically principled, approach for extending the model to obtain globalization. Of course the icRBM and icDBN models would allow globalization as well, but could be more difficult to work with[8].

Finally, we mention extensions related to boundary prediction assessment. A widely used metric in assessing image restoration quality is the structural similarity index (SSIM) [Wang et al., 2004] (see also Appendix A.5). This metric takes spatial context into account in the quality assessment. One possible avenue for future work on better evaluation methods is adapting methodologies within the SSIM metric or its extensions to the assessment of contour prediction quality. This quality could be defined in terms of the accuracy of contour probability map provided by the algorithm, compared to that obtained by taking the average of the human annotation maps, effectively producing a probability map. See

[8]for example due to the need for negative phase in training [Neal, 1992].

Appendix D.2 for further discussion, and experimental results related to using such approach.

Chapter 6

Conclusions and Future Work

6.1 Summary and Contributions

6.1.1 Transformation equivariant Boltzmann machines

In Chapter 3 we have developed a novel framework for obtaining Boltzmann machines and their deep extensions, in which the hidden unit activations co-transform with transformed input stimuli in a stable and predictable way, throughout the network. We define such models to be transformation equivariant. Such a property is clearly useful for computer vision systems, and has also been motivational, for example, in the development of steerable filters. Transformation equivariance is accomplished in our framework by encoding such properties into the network, by considering transformed versions of canonical-view parameters, and allowing hidden units to choose from the set of views.

Translation equivariant feature sharing (also known as convolutional feature sharing) has been the method for scaling image models (including those based on stochastic neural networks) beyond patches. In our framework we extend shallow and deep models to account for other kinds of transformations as well, focusing on in-plane rotations in the thesis and in Kivinen and Williams [2011]. The key idea is to associate each hidden unit with a latent transformation variable, which describes the selection of the transformed view of the canonical unit weights. In learning, only the canonical view weights need to be learned, as the transformations can be described by fixed geometrical transformation operators. Models based on the framework thus avoid having to learn transformed versions of the same patterns at all levels in the network, which is useful for

obtaining parsimonious and interpretable representations. The hidden units can be seen having an AND/OR-property with the unit activation variable applying the AND-operation, and the view-selection variable applying the OR-operation. AND/OR networks have been considered for example in Zhu and Mumford [2006], Zhu et al. [2008], and our approach can be seen as a principled way of having such nodes under a Boltzmann machine architecture.

6.1.2 Boltzmann machines for texture modelling

Our texture modelling work was motivated by the inability of state-of-the-art generative models to produce realistic natural images, including creating textured regions which are necessary subcomponents of any credible model of visual scenes, and complications in the quantitative assessment for such a highly variable class of data. In Chapter 4 we took a step back, and asked whether such models are able to generate textures, and what are the contributing factors to texture generation performance in terms of architectural properties. All of such models were conditionally Gaussian, and the analysis effectively achieved by focusing on the mPoT-model (Ranzato et. al., 2010), which has hidden units to control both the means and the precision matrices. To understand the differences we conducted a dissection by taking out hidden units of one of the two types, assessing the performance of the resulting models and the full model in the generation and inpainting of Brodatz-textures. The models used for these results covers the spectrum of Boltzmann machines typically used for image modelling. In the experiments we demonstrated performance comparable or better than state-of-the-art texture synthesis with the mPoT, which was nearly matched by the mean-only model. The results underlined the importance having hidden units to control the means of the components, under unconstrained synthesis.

We then considered structured extensions for modelling more complicated data, developing the multiple texture Boltzmann-machine framework. In this framework the joint probability density over texture images from several classes factorizes as a product of texture-class-specific probability distributions on the class-specific texture images, each defined by Boltzmann machines. The key idea is that only some of the parameters are class-specific, while the rest are shared. We showed that with the mean-unit-only model as the base model, and having only the hidden unit biases as the class-specific parameters, synthesis results were

obtained comparable to the individually-trained texture models which already provided state-of-the-art results. Importantly we then demonstrated empirically that the model provides means to generate images of differently textured regions. This is accomplished by considering the base model and the full set of learned parameters, and associating hidden unit biases at each position to those associated with the texture classes from which generation should happen there.

6.1.3 Texture interpolation Boltzmann machines

We have shown in section 4.6 that interpolating the hidden unit biases under our multiple texture models resulted in texture interpolation. This generated novel texture-type data, with smooth and abrupt transitioning effects between different texture types, according to the spatial interpolation pattern defined, in an effective manner. This was also shown for a single texture but under different view-points and illumination conditions, each defining their own classes. The feature sharing by these multi-texture models is expected to yield savings in terms of the number of features needed to model several categories, and provides a natural route for extension to a more comprehensive natural image model. It is also shown to address considering several computational photography problems under a single, principled statistical framework.

6.1.4 Multistream networks for visual boundary prediction

In Chapter 5 we considered contour detection, i.e. prediction of the presence of a visual boundary at a given image location. The motivation for the work was two-fold: Firstly, we wanted to improve understanding of what are the roles of the mean and the covariance hidden units of such conditionally Gaussian models, in the generation of image segment boundaries. Secondly, we wanted to investigate deep neural network architectures for learning the boundary detection problem, which is expected to be useful also for further generative model development.

We have developed and evaluated a range of neural network architectures for the task. Notable aspects of the work is (i) the use of "covariance features" [Ranzato and Hinton, 2010] which depend on the *squared* response of a filter to the input image, and (ii) the integration of image information from multiple scales and semantic levels via multiple streams of interlinked, layered, and non-linear "deep" processing in an end-to-end optimizable architecture.

Our results on the Berkeley Segmentation Data Set 500 (BSDS500) show comparable or better performance to the top-performing methods gPb [Arbelaez et al., 2011], SCG [Ren and Bo, 2012], and Sketch Tokens [Lim et al., 2013]. Additionally, our approach provides the fastest reported prediction times, and avoids several hand-engineered and/or computationally complex designs which are part of the the first two approaches cited above. Although the SCG [Ren and Bo, 2012] is more learning-based, both of these include hand-crafted oriented half-disc pooling, and also the use of computationally complex globalization which involves eigenvector calculations of matrices of the size of the input image (expected to scale cubically to the number of pixels, as opposed to linearly as in our approach). In our analysis, we provide a careful disection of the performance in terms of architecture, feature-types used and training methods, which provide clear signals for model understanding and further development.

6.2 Future Work

6.2.1 Extensions to the transformation equivariant modelling

The thesis has developed models equivariant with respect to translation and in-plane rotation. As discussed, a natural further equivariance to consider is scale. Without loss of generality, the transformations could be several other kinds of geometrical transformations. What is required is that they are fixed (linear) transformations of the input.

Considering additional local transformations (to rotation considered already in translation and rotation equivariant models in Kivinen and Williams [2011] and in Chapter 3) is another direction of extension as future work, but they might result in non-transformation equivariant models. Such an idea was explored by Sohn and Lee [2012]. Examples of such transformations include local affine transformations, so the transformational state variable would describe the view selection of a canonical feature transformed using a local affine transformation. A simple example of a different local transformation than what we have already considered would be a local shift. This could provide effective means for border handling, as the shifts would provide a means to control the amount of local constraints per visible lattice site. This approach would provide a data-adaptively-convolutional feature sharing method.

When used as a generative image model, the mcRBM typically produces Gabor-like covariance filters, as seen in Chapter 5. By making the associated hidden units equivariant by our framework, the models might yield benefits for learning and inference, including making the model easier to interpret. Building in further transformation equivariance onto the the contour prediction networks might result in similar benefits.

6.2.2 Extensions to the texture modelling algorithms

Here we discuss extensions to the multiple-texture model developed in Chapter 4. One direction is a hierarchical extension, in which the biases at each site, which affect the texture type to be generated there, depend on a higher layer hidden units; the simplest case is discrete random variables specifying the texture class. A different novel direction is placing a prior distribution on the fields of biases; this could be achieved using a gated MRF, which is good in modelling data discontinuities. The framework, including the prior on the biases, and the multiple-texture sampling function, would allow the generation of multiply-textured images, without the need to specify how many textures to generate, their spatial arrangement, texture gradients and so on. These setups have empirical support from the thesis experiments, where interpolation of the hidden unit biases under the multiple texture model is shown to allow interpolation of texture data. This includes the generation of novel texture data (outside the training textures), with both smooth and abrupt transitioning effects between different texture types.

Luo et al. [2013] demonstrates improved performance in generative texture modelling by using deep belief networks as opposed to shallow ones, based on a convolutional Spike-and-Slab RBM. In the Chapter 4 experiments, only shallow models were considered as the base models. However, the framework allows for considering deeper networks. Such a deep base model could be, for example, an mcDBN as used in Chapter 5 as a part of the contour prediction network, and the hidden unit biases (both mean and covariance) would be texture-class specific, and the rest shared across the different classes.

6.2.3 Extensions to the visual boundary prediction methods

Here we discuss extensions to the boundary prediction networks developed in Chapter 5. As discussed in the chapter, while some of the parameters were initialized by using generative pre-training, many of the parameters (the so-called read-out parameters) were initialized to random values. One thread for future work is building joint models of the image data and the boundary annotations (called icRBM, and icDBN models in Chapter 5). These models would contain the full set of parameters for the initialization of the boundary prediction network. Such joint models would allow for also other kinds of interesting applications, including image-prediction from boundary data (de-sketching) and image completion. For the latter task, boundary data might be available, and improve performance.

A different line of future work is involved with architectural extensions to the boundary prediction networks. These could include more-flexible stream integration models, and using transformation-equivariant covariance filters to provide direction-encoding. It is noted that also the boundary prediction networks could be extended for other image analysis tasks, such as image restoration.

Finally we mention extensions for obtaining "globalization" in the boundary prediction task, i.e. having a joint conditional distribution so that the predictions at each site are dependent on those at several other sites (and optimally all). As discussed in Chapter 5, using only deterministic units in the prediction networks means that decisions at the final layer are always done at each site independently of the other sites. Clearly the joint models discussed above would allow for globalization, but the fully generative setting might not work well in the discriminative application. Another statistically principled way to meet the goal would be to consider stochastic units in place of deterministic units in the prediction networks, possibly only in a few layers and feature planes. However, there would need to be stochastic units for which Markov blanket would include other spatially offset stochastic units for communication of the states, and thus enable globalization to happen[1]. For a similar performance level, it might be the case that the number of parameters in a deterministic network might become so large that combatting overfitting might become problematic. With the increased representational capacity would come increased computational complexity in learning

[1]The connections between the stochastic nodes could be made undirected in which case one would obtain a CRF-structure.

and inference. However, these models containing Bayesian network structures do not have a negative phase in training (as opposed to the training of the icRBM and icDBN models), which can be beneficial [Neal, 1992].

Appendix A

Background

A.1 Generic form of the log-likelihood gradient under a Boltzmann machine

Here we will derive the result in (2.35). The log-likelihood of training data assuming N training cases, conditional on model parameters θ can be written as follows

$$L(\theta) = \log p\big(\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(N)} \mid \theta\big) = -N \log Z + \sum_{n=1}^{N} \log \sum_{\mathbf{h}} \exp\big\{-\mathrm{E}\big(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta\big)\big\}. \tag{A.1}$$

Taking the derivative, we obtain that

$$\frac{\partial L(\theta)}{\partial \theta} = -N \frac{1}{Z} \frac{\partial Z}{\partial \theta} - \sum_{n=1}^{N} \frac{\sum_{\mathbf{h}} \exp\big\{-\mathrm{E}\big(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta\big)\big\} \frac{\partial \mathrm{E}\left(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta\right)}{\partial \theta}}{\sum_{\mathbf{h}} \exp\left\{-\mathrm{E}(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta)\right\}}. \tag{A.2}$$

Expanding the first term, and using cancelling terms $1/Z$ in the second term we obtain that

$$\begin{aligned}
\frac{\partial L(\theta)}{\partial \theta} &= N \frac{1}{Z} \sum_{\mathbf{v},\mathbf{h}} \exp\left\{-\mathrm{E}(\mathbf{v}, \mathbf{h} \mid \theta)\right\} \frac{\partial \mathrm{E}(\mathbf{v}, \mathbf{h} \mid \theta)}{\partial \theta} \\
&- \sum_{n=1}^{N} \frac{\frac{1}{Z} \sum_{\mathbf{h}} \exp\big\{-\mathrm{E}\big(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta\big)\big\} \frac{\partial \mathrm{E}\left(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta\right)}{\partial \theta}}{\frac{1}{Z} \sum_{\mathbf{h}} \exp\left\{-\mathrm{E}(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta)\right\}} \\
&= N \sum_{\mathbf{v},\mathbf{h}} \frac{1}{Z} \exp\left\{-\mathrm{E}(\mathbf{v}, \mathbf{h} \mid \theta)\right\} \frac{\partial \mathrm{E}(\mathbf{v}, \mathbf{h} \mid \theta)}{\partial \theta} \\
&- \sum_{n=1}^{N} \frac{\sum_{\mathbf{h}} \frac{1}{Z} \exp\big\{-\mathrm{E}\big(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta\big)\big\} \frac{\partial \mathrm{E}\left(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta\right)}{\partial \theta}}{\sum_{\mathbf{h}} \frac{1}{Z} \exp\left\{-\mathrm{E}(\mathbf{v}^{(n)}, \mathbf{h} \mid \theta)\right\}}.
\end{aligned} \tag{A.3}$$

Using definitions related to energy-based models, and those of conditional probabilities and the marginalization principle, we obtain that

$$\begin{aligned}
\frac{\partial L(\theta)}{\partial \theta} &= N \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h}|\theta) \frac{\partial \mathrm{E}(\mathbf{v},\mathbf{h} \mid \theta)}{\partial \theta} - \sum_{n=1}^{N} \frac{\sum_{\mathbf{h}} p\big(\mathbf{v}^{(n)},\mathbf{h} \mid \theta\big) \frac{\partial \mathrm{E}\left(\mathbf{v}^{(n)},\mathbf{h}|\theta\right)}{\partial \theta}}{p(\mathbf{v}^{(n)} \mid \theta)} \\
&= N \sum_{\mathbf{v},\mathbf{h}} p(\mathbf{v},\mathbf{h} \mid \theta) \frac{\partial \mathrm{E}(\mathbf{v},\mathbf{h} \mid \theta)}{\partial \theta} - \sum_{n=1}^{N} \sum_{\mathbf{h}} p\big(\mathbf{h} \mid \mathbf{v}^{(n)},\theta\big) \frac{\partial \mathrm{E}\big(\mathbf{v}^{(n)},\mathbf{h} \mid \theta\big)}{\partial \theta} \\
&= N \left\langle \frac{\partial \mathrm{E}(\mathbf{v},\mathbf{h} \mid \theta)}{\partial \theta} \right\rangle_{p(\mathbf{v},\mathbf{h}|\theta)} - \sum_{n=1}^{N} \left\langle \frac{\partial \mathrm{E}\big(\mathbf{v}^{(n)},\mathbf{h} \mid \theta\big)}{\partial \theta} \right\rangle_{p\left(\mathbf{h}|\mathbf{v}^{(n)},\theta\right)} \\
&= \sum_{n=1}^{N} \left(\left\langle \frac{\partial \mathrm{E}(\mathbf{v},\mathbf{h} \mid \theta)}{\partial \theta} \right\rangle_{p(\mathbf{v},\mathbf{h}|\theta)} - \left\langle \frac{\partial \mathrm{E}\big(\mathbf{v}^{(n)},\mathbf{h} \mid \theta\big)}{\partial \theta} \right\rangle_{p\left(\mathbf{h}|\mathbf{v}^{(n)},\theta\right)} \right).
\end{aligned}$$

A.2 Gradients in training a translation equivariant RBM

Here we consider training translation equivariant RBMs using an objective function consisting of two terms, a scaled negative data log-likelihood and a sparsity encouraging term, as in (2.50).

A.2.1 Log-likelihood gradient

Using the notation of (2.20), the log-probability of visible units $\mathbf{v}$ of a single training data case can be written as follows

$$\log p(\mathbf{v}|\theta) = -\log Z + a \sum_i v_i + \sum_k \sum_j \log \left(1 + \exp \left\{ b_k + \sum_{\ell \in \mathrm{N}_{kj}} v_\ell \omega^k_{d(j,\ell)} \right\} \right),$$

where the normalization constant

$$Z = \sum_{\mathbf{h}} \exp \left\{ \sum_k b_k \sum_j h^k_j \right\} \prod_i \left(1 + \exp \left\{ a + \sum_k \sum_{\ell \in \mathrm{N}(i)} h^k_\ell \omega^k_{d(\ell,i)} \right\} \right).$$

The partial derivatives of the log-likelihood with respect to the model parameters can be then written as follows:

$$\frac{\partial \log p(\mathbf{v} \mid \theta)}{\partial a} = -\frac{1}{Z}\frac{\partial Z}{\partial a} + \sum_i v_i \tag{A.4}$$

$$\frac{\partial \log p(\mathbf{v} \mid \theta)}{\partial b_k} = -\frac{1}{Z}\frac{\partial Z}{\partial b_k} + \sum_j \text{logistic}\left(b_k + \sum_{\ell \in \mathrm{N}_{kj}} v_\ell \omega^k_{d(j,\ell)}\right) \tag{A.5}$$

$$\frac{\partial \log p(\mathbf{v} \mid \theta)}{\partial \omega^k_h} = -\frac{1}{Z}\frac{\partial Z}{\partial \omega^k_h} + \sum_j v_{e(j,h)} \text{logistic}\left(b_k + \sum_{\ell \in \mathrm{N}_{kj}} v_\ell \omega^k_{d(j,\ell)}\right), \tag{A.6}$$

where $\text{logistic}(z) = \frac{1}{1+\exp(-z)}$ (which applies the logistic-function element-wise for vectors), and $e(j,h)$ is such that $d(j, e(j,h)) = h$. The partial derivatives of the log-normalization constant are obtained as:

$$\frac{1}{Z}\frac{\partial Z}{\partial b_k} = \sum_{\mathbf{h}} p(\mathbf{h}) \sum_j h^k_j \tag{A.7}$$

$$\frac{1}{Z}\frac{\partial Z}{\partial a} = \sum_{\mathbf{h}} p(\mathbf{h}) \sum_i \text{logistic}\left(a + \sum_k \left[\sum_{\ell \in \mathrm{N}(i)} h^k_\ell \omega^k_{d(\ell,i)}\right]\right) \tag{A.8}$$

$$\frac{1}{Z}\frac{\partial Z}{\partial \omega^k_h} = \sum_{\mathbf{h}} p(\mathbf{h}) \sum_{i'} h_{f(i',h)} \text{logistic}\left(a + \sum_k \left[\sum_{\ell \in \mathrm{N}(i')} h^k_\ell \omega^k_{d(\ell,i')}\right]\right), \tag{A.9}$$

where $f(i,h)$ is such that $d(f(i,h), i) = h$, and i' is such that $\sum_j \delta(d(j,i'), h) = 1$. Gathering the results, we obtain

$$\frac{\partial \log p(\mathbf{v} \mid \theta)}{\partial a} = \sum_i v_i - \sum_{\mathbf{h}} p(\mathbf{h}) \sum_i \text{logistic}\left(a + \sum_k \left[\sum_{\ell \in \mathrm{N}(i)} h^k_\ell \omega^k_{d(\ell,i)}\right]\right) \tag{A.10}$$

$$\frac{\partial \log p(\mathbf{v} \mid \theta)}{\partial b_k} = \sum_j \text{logistic}\left(b_k + \sum_{\ell \in \mathrm{N}_{kj}} v_\ell \omega^k_{d(j,\ell)}\right) - \sum_{\mathbf{h}} p(\mathbf{h}) \sum_j h^k_j \tag{A.11}$$

$$\begin{aligned}\frac{\partial \log p(\mathbf{v} \mid \theta)}{\partial \omega^k_h} = &\sum_j v_{e(j,h)} \text{logistic}\left(b_k + \sum_{\ell \in \mathrm{N}_{kj}} v_\ell \omega^k_{d(j,\ell)}\right) \\ &- \sum_{\mathbf{h}} p(\mathbf{h}) \sum_{i'} h_{f(i',h)} \text{logistic}\left(a + \sum_k \left[\sum_{\ell \in \mathrm{N}(i')} h^k_\ell \omega^k_{d(\ell,i')}\right]\right).\end{aligned} \tag{A.12}$$

For N training data cases, the partial derivatives become:

$$\frac{\partial \log p\big(\mathbf{v}^{(1)},\ldots,\mathbf{v}^{(N)} \mid \theta\big)}{\partial a} = \sum_{n=1}^{N}\sum_{i} v_i^{(n)}$$
$$- N\sum_{\mathbf{h}} p(\mathbf{h})\sum_{i}\text{logistic}\left(a + \sum_{k}\left[\sum_{\ell\in \mathrm{N}(i)} h_\ell^k \omega_{d(\ell,i)}^k\right]\right) \tag{A.13}$$
$$\frac{\partial \log p\big(\mathbf{v}^{(1)},\ldots,\mathbf{v}^{(N)} \mid \theta\big)}{\partial b_k} = \sum_{n=1}^{N}\sum_{j}\text{logistic}\left(b_k + \sum_{\ell\in \mathrm{N}_{kj}} v_\ell^{(n)}\omega_{d(j,\ell)}^k\right)$$
$$- N\sum_{\mathbf{h}} p(\mathbf{h})\sum_{j} h_j^k \tag{A.14}$$
$$\frac{\partial \log p\big(\mathbf{v}^{(1)},\ldots,\mathbf{v}^{(N)} \mid \theta\big)}{\partial \omega_h^k} = \sum_{n=1}^{N}\sum_{j} v_{e(j,h)}\text{logistic}\left(b_k + \sum_{\ell\in \mathrm{N}_{kj}} v_\ell^{(n)}\omega_{d(j,\ell)}^k\right)$$
$$- N\sum_{\mathbf{h}} p(\mathbf{h})\sum_{i'} h_{f(i',h)}\text{logistic}\left(a + \sum_{k}\left[\sum_{\ell\in \mathrm{N}(i')} h_\ell^k \omega_{d(\ell,i')}^k\right]\right). \tag{A.15}$$

A.2.2 Sparsity regularization term gradient

Let us assume

$$p(h_j^{k(n)} = 1|\mathbf{v}^{(n)},\theta) = \text{logistic}\left\{b_s^k + \sum_{\ell\in \mathrm{N}_{kj}} v_\ell^{(n)}\omega_{d(j,\ell)}^k\right\},$$

and thus gradient of the sparsity regularization term is

$$\frac{\partial p(h_j^{k(n)} = 1|\mathbf{v}^{(n)},\theta)}{\partial \theta^k} = p(h_j^{k(n)} = 1|\mathbf{v}^{(n)},\theta)$$
$$\cdot\Big(1 - p(h_j^{k(n)} = 1|\mathbf{v}^{(n)},\theta)\Big)\frac{\partial}{\partial\theta^k}\left\{b_s^k + \sum_{\ell\in \mathrm{N}_{kj}} v_\ell^{(n)}\omega_{d(j,\ell)}^k\right\}. \tag{A.16}$$

We can then find that

$$\frac{\partial p(h_j^{k(n)} = 1|\mathbf{v}^{(n)},\theta)}{\partial b_s^k} = \text{logistic}\left\{b_s^k + \sum_{\ell\in \mathrm{N}_{kj}} v_\ell^{(n)}\omega_{d(j,\ell)}^k\right\}$$
$$\cdot\left(1 - \text{logistic}\left\{b_s^k + \sum_{\ell\in \mathrm{N}_{kj}} v_\ell^{(n)}\omega_{d(j,\ell)}^k\right\}\right), \tag{A.17}$$

and similarly

$$\frac{\partial p(h_j^{k(n)}=1|\mathbf{v}^{(n)},\theta)}{\partial \omega_h^k} = \text{logistic}\left\{b_s^k + \sum_{\ell\in \mathrm{N}_{kj}} v_\ell^{(n)}\omega_{d(j,\ell)}^k\right\} \cdot \left(1-\text{logistic}\left\{b_s^k + \sum_{\ell\in \mathrm{N}_{kj}} v_\ell^{(n)}\omega_{d(j,\ell)}^k\right\}\right) v_{e(j,h)}, \quad \text{(A.18)}$$

where (as above) $e(j,h)$ is such that $d(j,e(j,h))=h$.

A.3 Marginalizing hidden units under PoT

We will start by deriving the free-energy under a PoT from its hidden unit expanded form, using the following relation [Osindero et al., 2006]

$$\frac{\Gamma\left(\alpha+\frac{1}{2}\right)}{\Gamma(\alpha)\sqrt{2\pi}}\left(1+\frac{1}{2}\tau^2\right)^{-\left(\alpha+\frac{1}{2}\right)} = \int d\lambda \left(\frac{1}{\sqrt{2\pi}}\lambda^{\frac{1}{2}}\exp\left\{-\frac{1}{2}\tau^2\lambda\right\}\right)\left(\frac{1}{\Gamma(\alpha)}\lambda^{\alpha-1}\exp\{-\lambda\}\right). \quad \text{(A.19)}$$

Taking the logarithm of both sides we obtain that

$$\log\left\{\frac{\Gamma\left(\alpha+\frac{1}{2}\right)}{\Gamma(\alpha)\sqrt{2\pi}}\right\} - \left(\alpha+\frac{1}{2}\right)\log\left\{1+\frac{1}{2}\tau^2\right\} = \log\left\{\int d\lambda \exp\left\{-\lambda\left(1+\frac{1}{2}\tau^2\right)\right\}\right\} + \log\left\{\frac{1}{\Gamma(\alpha)\sqrt{2\pi}}\right\}. \quad \text{(A.20)}$$

Defining $\beta = \alpha + \frac{1}{2}$, and simplifying we obtain that

$$\log\{\Gamma(\beta)\} - \beta\log\left\{1+\frac{1}{2}\tau^2\right\} = \log\left\{\int \lambda^{\beta-1}\exp\left\{-\lambda\left(1+\frac{1}{2}\tau^2\right)\right\}d\lambda\right\} \quad \text{(A.21)}$$

By definition, the free-energy of the PoT with continuous auxiliary hidden variables $\mathbf{h}$ can be written as

$$F_{\mathrm{PoT}}(\mathbf{v}|\theta) = -\log\left\{\int_{\mathcal{H}} d\mathbf{h}\exp\{-E_{\mathrm{PoT}}(\mathbf{h},\mathbf{v}|\theta)\}\right\}. \quad \text{(A.22)}$$

Let us define

$$E_{\mathrm{PoT}}(\mathbf{v},\mathbf{h}|\theta) = -\sum_j\left[(\gamma_j-1)\log h_j - h_j\left(1+\frac{1}{2}\left[\mathrm{C}_{\cdot j}{}^{\top}\mathbf{v}\right]^2\right) - \log\{\Gamma(\gamma_j)\}\right]. \quad \text{(A.23)}$$

Then we obtain that

$$F_{\mathrm{PoT}}(\mathbf{v}|\theta) = \tag{A.24}$$

$$-\log\left\{\int_{\mathcal{H}} d\mathbf{h}\exp\left\{\sum_j\left[(\gamma_j-1)\log h_j - h_j\left(1+\frac{1}{2}\left[\mathrm{C}_{\cdot j}{}^{\top}\mathbf{v}\right]^2\right)-\log\{\Gamma(\gamma_j)\}\right]\right\}\right\} \tag{A.25}$$

$$=\sum_j\log\{\Gamma(\gamma_j)\}-\log\left\{\prod_j\int dh_j\exp\left\{(\gamma_j-1)\log h_j - h_j\left(1+\frac{1}{2}\left[\mathrm{C}_{\cdot j}{}^{\top}\mathbf{v}\right]^2\right)\right\}\right\} \tag{A.26}$$

$$=\sum_j\log\{\Gamma(\gamma_j)\}-\sum_j\log\left\{\int dh_j\exp\left\{(\gamma_j-1)\log h_j - h_j\left(1+\frac{1}{2}\left[\mathrm{C}_{\cdot j}{}^{\top}\mathbf{v}\right]^2\right)\right\}\right\}. \tag{A.27}$$

Using the result in (A.21) we obtain that

$$F_{\mathrm{PoT}}(\mathbf{v}|\theta) = \tag{A.28}$$

$$\sum_j\log\{\Gamma(\gamma_j)\}-\sum_j\log\left\{\int dh_j\left[h_j^{(\gamma_j-1)}\exp\left\{-h_j\left(1+\frac{1}{2}\left[\mathrm{C}_{\cdot j}{}^{\top}\mathbf{v}\right]^2\right)\right\}\right]\right\} \tag{A.29}$$

$$=\sum_j\log\{\Gamma(\gamma_j)\}+\sum_j\left[\gamma_j\log\left\{1+\frac{1}{2}\left[\mathrm{C}_{\cdot j}{}^{\top}\mathbf{v}\right]^2\right\}-\log\{\Gamma(\gamma_j)\}\right] \tag{A.30}$$

$$=\sum_j\gamma_j\log\left\{1+\frac{1}{2}\left[\mathrm{C}_{\cdot j}{}^{\top}\mathbf{v}\right]^2\right\}. \tag{A.31}$$

Note that by defining

$$E_{\mathrm{PoT}}(\mathbf{v},\mathbf{h}|\theta)=-\sum_j\left[(\gamma_j-1)\log h_j - h_j\left(1+\frac{1}{2}\left[\mathrm{C}_{\cdot j}{}^{\top}\mathbf{v}\right]^2\right)\right], \tag{A.32}$$

one would obtain that

$$F_{\mathrm{PoT}}(\mathbf{v}|\theta)=\sum_j\left[\gamma_j\log\left\{1+\frac{1}{2}\left[\mathrm{C}_{\cdot j}{}^{\top}\mathbf{v}\right]^2\right\}-\log\{\Gamma(\gamma_j)\}\right]. \tag{A.33}$$

A.4 Experiment: Modeling 1.5D data with a TE-RBM

In this experiment, we explored the problem of modeling binary data of 1.5D structure with a binary-unit translation-equivariant RBM (TE-RBM). As shown

in Figure A.1(d), the data consisted of an exhaustive set of spatially shifted versions of a canonical image, containing a single spatially contiguous segment of ones (referred to as a bar) on otherwise zero data (background).

One of the main goals of this experiment was to explore the capability boundaries of the parameter-reduced model, and the required model complexity in the context of this binary-data modeling problem. Another main goal was to investigate the possibility of learning without imposing discriminative learning terms, and the properties of a TE-RBM model, which would be a good generative model, but would produce sparse activation patterns in response to the data. The property of sparsity is desirable with deep belief networks for example in terms of interpretability of inferences, and the encouragement of sparsity has been reported to be relatively essential in practice to learn features resembling the receptive fields of the primary visual cortex [Lee et al., 2008].

The TE-RBM that was used in the experiments contained a single hidden unit layer with same number of units as in the visible layer (13), a single bias for each of the hidden and visible layers, and connection weights wrapping around toroidally as in the TE-RBM of Figure 2.2(b). To learn model parameters we optimized a cost function containing log-likelihood of data under the model, and a sparsity encouragement term, as in (2.50). The log-probability of visible units in a single graph under this model can be written as follows:

$$\log p(\mathbf{v}|\theta) = -\log Z + a\sum_i v_i + \sum_k \sum_j \log\left(1 + \exp\left\{b_k + \sum_{\ell \in \mathrm{N}_{kj}} v_\ell \omega^k_{d(j,\ell)}\right\}\right), \tag{A.34}$$

where the normalization constant

$$Z = \sum_h \exp\left\{\sum_k b_k \sum_j h^k_j\right\} \prod_i \left(1 + \exp\left\{a + \sum_k \sum_{\ell \in \mathrm{N}(i)} h^k_\ell \omega^k_{d(\ell,i)}\right\}\right). \tag{A.35}$$

As the model contained so few units, computation of the exact gradient was possible. The learning was then done by using a scaled conjugate gradient (SCG) method [1]. The partial derivatives of the cost function with respect to model parameters (needed in the SCG-algorithm) are given in Appendix A.2.1, and A.2.2.

To study the flexibility and sparsity properties of the model, the receptive field size and the regularization constant λ of the sparsity encouragement cost function term were varied. Target activation rate was set to 1/13. Due to the

[1] The scg-function of the Netlab-toolbox version 3.3 was used.

existence of multiple local optima, learning was done by using multiple starts, each with different small random weights and biases. Figure A.1 shows results from learning assuming sparsity regularization constant $\lambda = 1$ and receptive field width 11, which was the narrowest possible to obtain optimal log-likelihood and sparsity rate assuming $\lambda = 1$. Figure A.2 shows most likely unit joint states assuming the model parameters, which confirm them good for the problem, since 13 most likely joint visibles are equal to the training data instances, and 13 most likely joint hidden unit states contain a single active unit, and the other configurations for both of these cases are much less probable. To summarize the findings, learning the model optimally with respect to the log-likelihood[2] required that the receptive field size was sufficiently large, in any case larger than the bar. In order to reach the target activation level as well, the regularization rate needed to be within a range of values; for too small values, the activation level was not met, and for too large values, the activation level was obtained but by having many of the hidden units weakly turned on.

A.5 Image Quality Assessment with the Structural Similarity Index (SSIM)

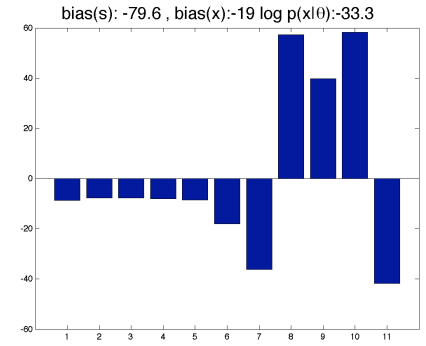
The structural similarity metric [Wang et al., 2004] assesses the similarity of two signals $\mathbf{x}$ and $\mathbf{y}$ of same dimension with J elements, using a function on three comparison functions which (aim to) measure differences in signal luminance contrast, and structure:

$$\mathrm{SSIM}(\mathbf{x}, \mathbf{y}) = [\ell(\mathbf{x}, \mathbf{y})]^{\alpha} \cdot [c(\mathbf{x}, \mathbf{y})]^{\beta} \cdot [s(\mathbf{x}, \mathbf{y})]^{\gamma}, \tag{A.36}$$

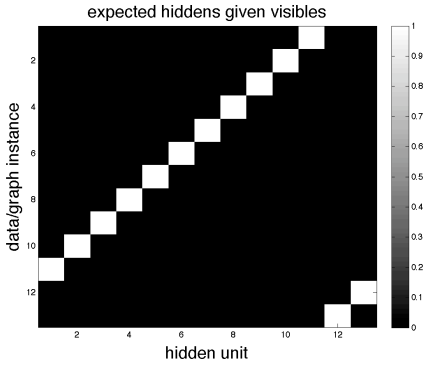
where the signal luminance comparison function $\ell(\mathbf{x}, \mathbf{y})$ operates on signal sample means ($\mu_z = \frac{1}{J}\sum_{j=1}^{J} z_j$), the contrast comparison function $c(\mathbf{x}, \mathbf{y})$ operates on signal standard deviations (variance $\sigma_z^2 = \frac{1}{J-1}\sum_{j=1}^{J}(z_j - \mu_z)^2$), the structure comparison function $s(\mathbf{x}, \mathbf{y})$ is a correlation-based measure between the signals, and $\alpha > 0$, $\beta > 0$ and $\gamma > 0$ denote scalars to adjust the relative importance of the three components.

The thesis uses the form implemented in Wang et al. [2004] in which $\alpha = \beta =$

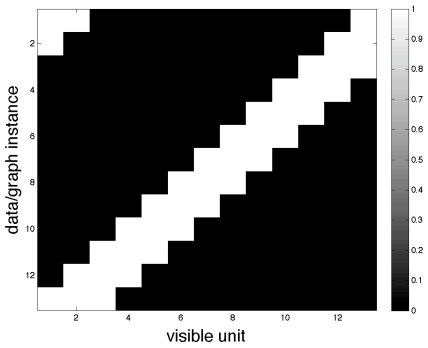
[2] As there are 13 data instances each having one of the 13 possible equally occurring patters, the optimal log-likelihood is $13 \log(1/13) = -33.3443$.



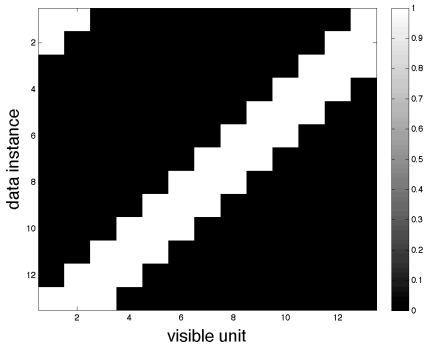
(a) Model parameters



(b) Expected hidden units conditional on data

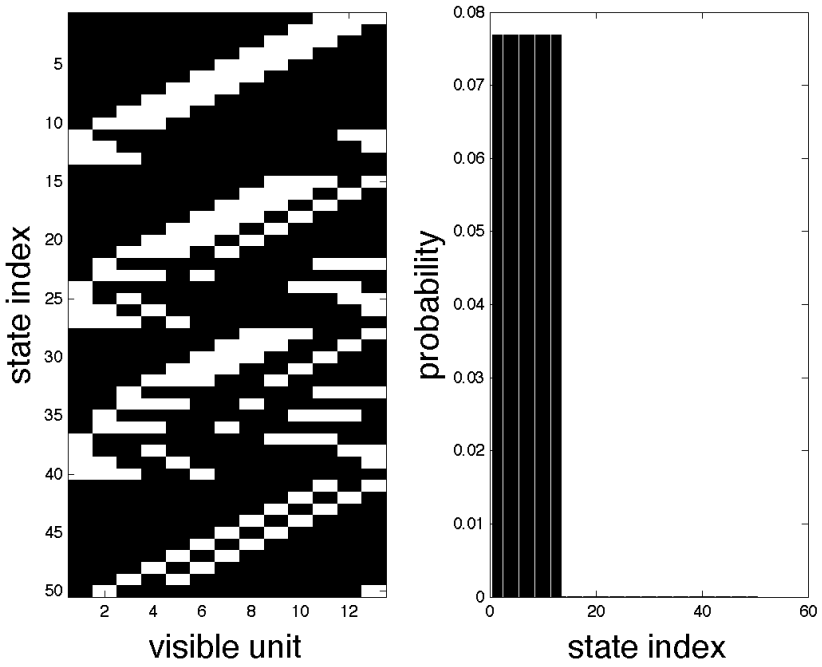


(c) Reconstruction

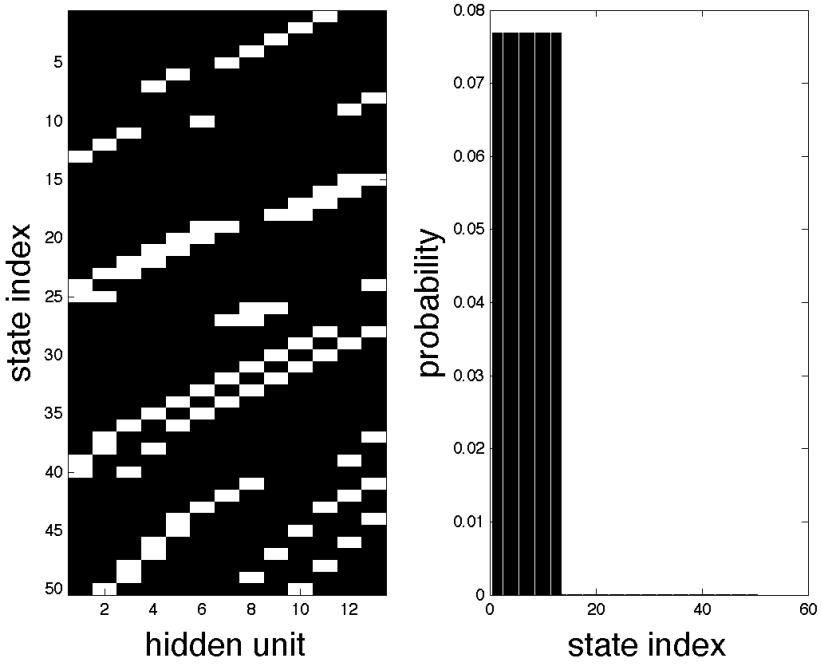


(d) Training data

Figure A.1: Results from learning a TE-RBM with receptive field width 11, assuming sparsity regularization constant $\lambda = 1$ from circularly shifting binary data shown in A.1(d). (a) shows parameters of one of the TE-RBM models, with shared best objective function value, and conditional on the parameters, (b) shows expected hidden units given the data, (c) shows reconstruction by computing expected visible units conditional on expected hidden units given data (which is basically perfect). Note that both the log-likelihood and sparsity are optimal given the model parameters.



(a) 50 most likely visible unit states and their probabilities.



(b) 50 most likely hidden unit states and their probabilities.

Figure A.2: Most likely joint states assuming model parameters as in A.1(a). Because the model is translation equivariant, states differing by translation are equally probable. From (a) we can see that the first 13 most likely visible joint states form the training data instances, and the other joint configurations are highly unlikely in comparison. From (b) we can see that the 13 most likely hidden joint states contain only one active unit, and the other joint configurations are highly unlikely in comparison.

$\gamma = 1$, and the specific form of the SSIM index is the following:

$$\mathrm{SSIM}(\mathbf{x}, \mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{\left(\mu_x^2 + \mu_y^2 + C_1\right)\left(\sigma_x^2 + \sigma_y^2 + C_2\right)}, \tag{A.37}$$

where $\sigma_{xy} = \frac{1}{J-1}\sum_{j=1}^{J}(x_j - \mu_x)(y_j - \mu_y)$, and C_1 and C_2 are scalars dependent on the dynamic range of the signals.

For image quality assessment Wang et al. [2004] recommends to apply the index locally (within local windows) and average the local quality assessments for an overall quality measure, called the mean SSIM (MSSIM) index:

$$\mathrm{MSSIM} = \frac{1}{I}\sum_{i=1}^{I}\hat{\mathrm{SSIM}}(\mathbf{x}_{\mathcal{N}_j}, \mathbf{y}_{\mathcal{N}_j}), \tag{A.38}$$

where $\mathbf{z}_{\mathcal{N}_j}$ denotes a local (2D) window within an image, I denotes the number of local windows in the image, and $\hat{\mathrm{SSIM}}(\mathbf{x}_{\mathcal{N}_j}, \mathbf{y}_{\mathcal{N}_j})$ is an evaluation of a modification of (A.37) to avoid "blocking" artifacts. In particular, the local SSIM evaluations are smoothed with a circularly symmetric Gaussian weighting function $\mathbf{w}$. The modified evaluations can be then seen to apply (A.37) but assume $\mu_z = \sum_{j=1}^{J} w_j z_j$, $\sigma_z = \left(\frac{1}{J-1}\sum_{j=1}^{J} w_j (z_j - \mu_z)^2\right)^{\frac{1}{2}}$, and $\sigma_{xy} = \sum_{j=1}^{J} w_j (x_j - \mu_x)(y_j - \mu_y)$.

The thesis uses the same settings (considered default/standard) in the evaluation as in Wang et al. [2004], with $C_1 = (0.01 \cdot 255)^2$, $C_2 = (0.03 \cdot 255)^2$ (the dynamic range of images assumed to be 255), and the Gaussian smoothing filter is of 11×11 size, assumes standard deviation of 1.5 samples, and is normalized to unit sum.

Appendix B

Transformation Equivariant Modelling

B.1 In-plane image rotations

For the in-plane rotations we consider, images consisting of 2D grids of pixels with locations x (with size $N \times 2$ specifying the N Cartesian coordinates of the grid) are rotated (counter-clockwise) about their center $\hat{x}$ with a certain rotation angle θ (in radians), with a geometric rotation matrix

$$T(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

to produce a rotated grid of pixels with locations $y = T(\theta)(x - \hat{x}) + \hat{x}$. We call the grid of pixels in the output image as the measurement grid. For rotation angles of multiples of 90 degrees, the grid of rotated pixels and the measurement grid align, and there exists a one-to-one mapping between the grid values. For other rotation angles, the grids do not align, and interpolation is necessary to faithfully obtain pixel values at the measurement grid locations[1]. See Figure B.1 for an illustration of in-plane rotations for both of these cases, in which the top-most dotted grid of pixels in the figure is rotated about its center 45, and 90 degrees. Our implementation uses bilinear interpolation, each measurement grid pixel evaluated as a convex combination of (maximally) four neighboring pixels of the rotated grid. Each of the weights is computed as a product of two terms which depend on the horizontal and vertical distance of the measurement grid

[1] with the exception of the center pixel

pixel with the rotated grid, to the pixels which are used in the interpolation. Denoting a_1 and a_2 as the (horizontal and vertical) coordinates of the measurement grid pixel within the rotated grid, b_1 and b_2 those of a pixel to which interpolation weight w is computed, then $w = (1 - |a_1 - b_1|)(1 - |a_2 - b_2|)$[2]. To compute the location of the measurement grid pixels within the coordinate system of the rotated grid, each measurement grid pixel is rotated with negative rotation angle $-\theta$. These locations are then used to find the neighboring units, and their associated interpolation weights. Note that the measurement grid also needs to be larger than the original image grid (or the grid of pixels we care about), since for some rotation angles the rotated grid positions extend outside of a grid of size of the original image, with the corners tracing out a bounding circle for a full range of rotations.

Note that $T(\theta)^{-1} = T(\theta)^\top = T(-\theta)$. The transformation matrix (consisting of interpolation weights / mapping coefficients), that is used to rotate images with rotation angle $a_k = 360(k-1)/K$ degrees, is denoted by $\mathbf{R}^{(k)}$, where k is an integer ranging from 1 to K. If no interpolation would be necessary it would hold that transforming an image with $[\mathbf{R}^{(k)}]^\top$ would equal transforming the image with transformation matrix that rotates an image by $-a_k$ or equivalently by $360 - a_k$ degrees. However, non-multiples of 90 degree rotations need interpolation, and differences may arise.

B.2 Conditional distributions related to the convolutional STEER-RBM

Here we will derive the conditional distributions in Section 3.3. As mentioned in 3.2.2, the energy for a convolutional STEER-RBM can be written as follows:

$$\mathrm{E}(\mathbf{v}, \mathbf{h}, \mathbf{z} \mid \theta) = \frac{1}{2\sigma^2} \sum_i \left(v_i^2 - 2a v_i\right) - \sum_\alpha \sum_j h_j^\alpha \left(b_\alpha + \frac{1}{\sigma} \sum_k z_{jk}^\alpha \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell), k) \right), \tag{B.1}$$

where a is visible unit layer bias, b_α is a bias for hidden unit feature plane α, $\mathrm{N}_{\alpha j}$ indexes the visible units within the receptive field of hidden unit h_j^α, and σ defines the standard deviation of the univariate Gaussian conditional distribution

[2] these weights are then normalized to sum to one, when interpolation has to be done from less than 4 pixels

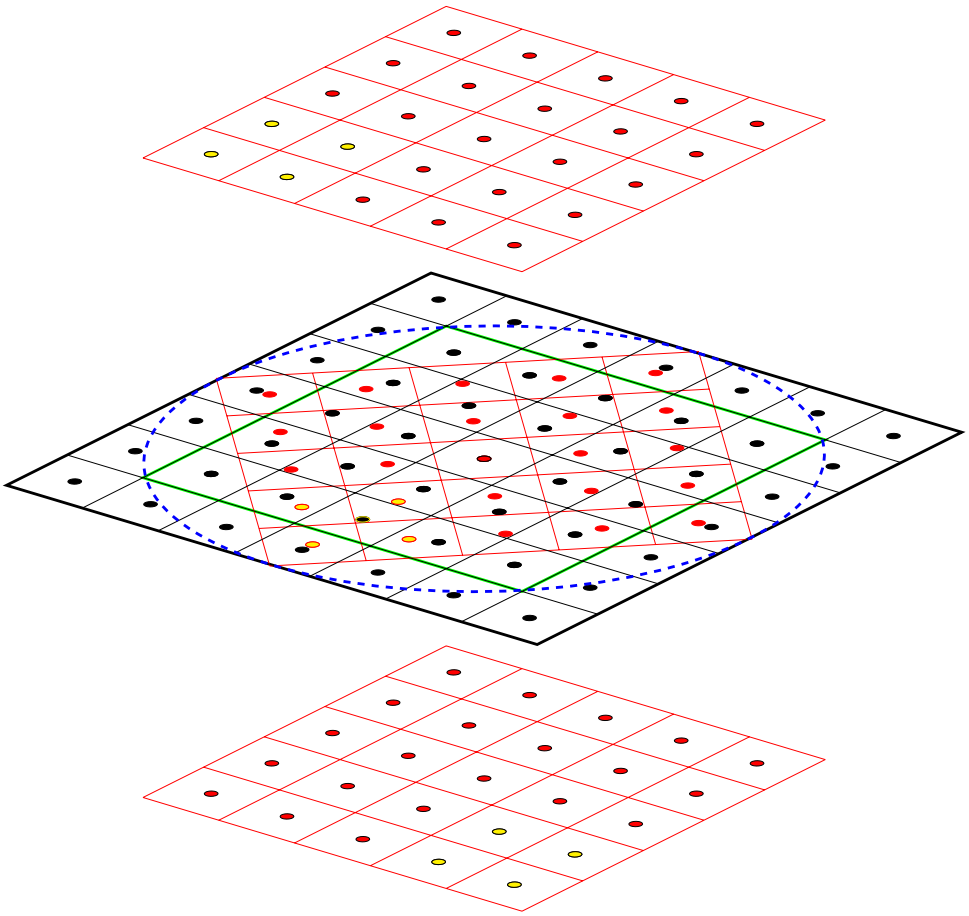


Figure B.1: An illustration of in-plane rotations of the top-most dotted grid of pixels about its center. The bottom grid shows an example of a 90 , and the middle 45, degree rotation counterclockwise. In the latter case, the rotated grid of pixels is overlaid on the measurement grid whose pixels are marked as black-filled circles, and the dimensions of the original grid is overlaid as a green square. In this case, computation of most of the measurement grid pixels requires interpolation. And for example the value of the pixel marked as a black-filled circle with yellow circumference is computed as a convex combination of the neighboring four yellow-filled circles in the rotated grid. The dashed blue circle in the middle grid depicts the circle that the corners of the rotating grid trace out.

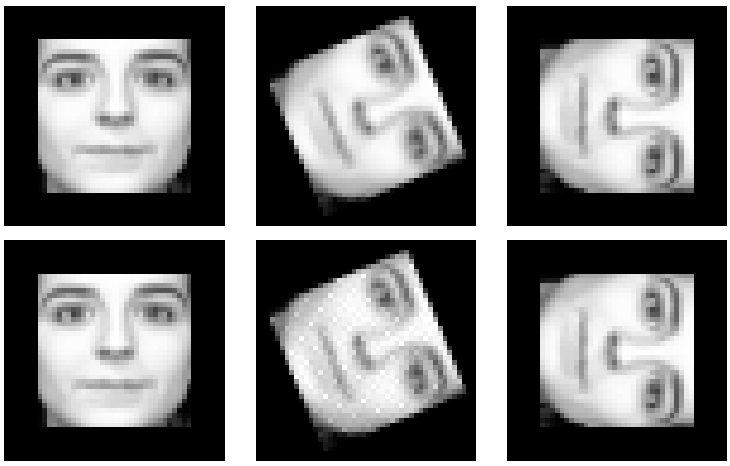


Figure B.2: Comparison of transforming an image (left-most images) by applying a transpose of a rotation matrix (images at the bottom row), and by applying a rotation matrix for inverted rotation angle (images at the top row). For multiples of 90 degrees they are exactly the same.

$p(v_i \mid \mathbf{h}, \mathbf{z}, \theta)$, as will be shown in the following.

Let us start by computing the joint marginal distribution of the visible units, first marginalizing out the hidden units:

$$
\begin{aligned}
p(\mathbf{v}, \mathbf{z} \mid \theta) &\triangleq \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}, \mathbf{z} \mid \theta) = \sum_{\mathbf{h}} \frac{1}{Z} \exp\left\{-\mathrm{E}(\mathbf{v}, \mathbf{h}, \mathbf{z} \mid \theta)\right\} \\
&= \frac{1}{Z} \exp\left\{-\frac{1}{2\sigma^2} \sum_i \left(v_i^2 - 2a v_i\right)\right\} \sum_{\mathbf{h}} \prod_{\alpha, j} \exp\left\{h_j^\alpha \left(b_\alpha + \frac{1}{\sigma} \sum_k z_{jk}^\alpha \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell), k)\right)\right\} \\
&= \frac{1}{Z} \exp\left\{-\frac{1}{2\sigma^2} \sum_i \left(v_i^2 - 2a v_i\right)\right\} \prod_{\alpha, j} \sum_{h_j^\alpha} \exp\left\{h_j^\alpha \left(b_\alpha + \frac{1}{\sigma} \sum_k z_{jk}^\alpha \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell), k)\right)\right\} \\
&= \frac{1}{Z} \exp\left\{-\frac{1}{2\sigma^2} \sum_i \left(v_i^2 - 2a v_i\right)\right\} \prod_{\alpha, j} \left(1 + \exp\left\{b_\alpha + \frac{1}{\sigma} \sum_k z_{jk}^\alpha \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell), k)\right\}\right),
\end{aligned}
$$

where the second-last equality follows from using the technique of 'pushing sums into products' (the exponential terms on the right of previous line contain only one hidden unit, and so the sums can be pushed into the products). We then marginalize the rotation assignment variables to get the joint marginal of the

visible units, using similar reasoning:

$$
\begin{aligned}
&p(\mathbf{v} \mid \theta) \triangleq \sum_{\mathbf{z}} p(\mathbf{v}, \mathbf{z} \mid \theta) \\
&= \frac{1}{Z} \exp\left\{-\frac{1}{2\sigma^2}\sum_i \left(v_i^2 - 2av_i\right)\right\} \sum_{\mathbf{z}} \prod_{\alpha,j} \left(1 + \exp\left\{b_\alpha + \frac{1}{\sigma}\sum_k z_{jk}^\alpha \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right\}\right) \\
&= \frac{1}{Z} \exp\left\{-\frac{1}{2\sigma^2}\sum_i \left(v_i^2 - 2av_i\right)\right\} \prod_{\alpha,j} \sum_{\mathbf{z}_j^\alpha} \left(1 + \exp\left\{b_\alpha + \frac{1}{\sigma}\sum_k z_{jk}^\alpha \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right\}\right) \\
&= \frac{1}{Z} \exp\left\{-\frac{1}{2\sigma^2}\sum_i \left(v_i^2 - 2av_i\right)\right\} \prod_{\alpha,j} \left(K + \sum_{k=1}^K \exp\left\{b_\alpha + \frac{1}{\sigma}\sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right\}\right).
\end{aligned}
$$

Proceeding on deriving the partial derivatives in Section 3.3, we then compute

$$
\begin{aligned}
&p(\mathbf{h}, \mathbf{z} \mid \mathbf{v}, \theta) \triangleq \frac{p(\mathbf{v}, \mathbf{h}, \mathbf{z} \mid \theta)}{p(\mathbf{v} \mid \theta)} \\
&= \frac{\frac{1}{Z}\exp\left\{-\frac{1}{2\sigma^2}\sum_i \left(v_i^2 - 2av_i\right)\right\} \prod_{\alpha,j} \exp\left\{h_j^\alpha \left(b_\alpha + \frac{1}{\sigma}\sum_k z_{jk}^\alpha \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right)\right\}}{\frac{1}{Z}\exp\left\{-\frac{1}{2\sigma^2}\sum_i \left(v_i^2 - 2av_i\right)\right\} \prod_{\alpha,j} \left(K + \sum_{k=1}^K \exp\left\{b_\alpha + \frac{1}{\sigma}\sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right\}\right)} \\
&= \prod_{\alpha,j} p\big(h_j^\alpha, \mathbf{z}_j^\alpha \mid \mathbf{v}, \theta\big),
\end{aligned}
$$

where

$$
p\big(h_j^\alpha, \mathbf{z}_j^\alpha \mid \mathbf{v}, \theta\big) = \frac{\exp\left\{h_j^\alpha \left(b_\alpha + \frac{1}{\sigma}\sum_k z_{jk}^\alpha \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right)\right\}}{K + \sum_{k=1}^K \exp\left\{b_\alpha + \frac{1}{\sigma}\sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right\}}. \tag{B.2}
$$

Thus we obtain that

$$
p\big(h_j^\alpha \mid \mathbf{v}, \theta\big) = \sum_{\mathbf{z}_j^\alpha} p\big(h_j^\alpha, \mathbf{z}_j^\alpha \mid \mathbf{v}, \theta\big) = \frac{\sum_{k=1}^K \exp\left\{h_j^\alpha \left(b_\alpha + \frac{1}{\sigma}\sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right)\right\}}{K + \sum_{k=1}^K \exp\left\{b_\alpha + \frac{1}{\sigma}\sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right\}}, \tag{B.3}
$$

as in (3.6). This can be then used to compute the 'remaining' conditional distribution

$$
p\big(\mathbf{z}_j^\alpha \mid h_j^\alpha, \mathbf{v}, \theta\big) \triangleq \frac{p\big(h_j^\alpha, \mathbf{z}_j^\alpha \mid \mathbf{v}, \theta\big)}{p\big(h_j^\alpha \mid \mathbf{v}, \theta\big)} = \frac{\exp\left\{h_j^\alpha \left(b_\alpha + \sum_{k=1}^K z_{jk}^\alpha \frac{1}{\sigma}\sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right)\right\}}{\sum_{k=1}^K \exp\left\{h_j^\alpha \left(b_\alpha + \frac{1}{\sigma}\sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right)\right\}},
$$

which is equal to (3.7). An alternative factorization of the joint conditional of the hidden units $p(\mathbf{h}, \mathbf{z} \mid \mathbf{v}, \theta)$ to those shown in (3.6) and (3.7) is given below,

but using them in sampling is computationally more wasteful (in theory)[3]

$$p\big(\mathbf{z}_j^\alpha \mid \mathbf{v}, \theta\big) = \frac{1 + \exp\left\{b_\alpha + \sum_{k=1}^K z_{jk}^\alpha \frac{1}{\sigma} \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right\}}{K + \sum_{k=1}^K \exp\left\{b_\alpha + \frac{1}{\sigma} \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right\}} \tag{B.4}$$

$$p\big(h_j^\alpha \mid \mathbf{z}_j^\alpha, \mathbf{v}, \theta\big) = \frac{\exp\left\{h_j^\alpha \left(b_\alpha + \sum_{k=1}^K z_{jk}^\alpha \frac{1}{\sigma} \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right)\right\}}{1 + \exp\left\{b_\alpha + \sum_{k=1}^K z_{jk}^\alpha \frac{1}{\sigma} \sum_{\ell \in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\right\}}. \tag{B.5}$$

To derive the full conditional distribution of a visible unit, we note that

$$\begin{aligned}
p(\mathbf{v} \mid \mathbf{h}, \mathbf{z}, \theta) &\triangleq \frac{p(\mathbf{v}, \mathbf{h}, \mathbf{z} \mid \theta)}{p(\mathbf{h}, \mathbf{z} \mid \theta)} = \\
&\frac{\frac{1}{Z} \exp\left\{\sum_{\alpha,j} b_\alpha h_j^\alpha\right\} \prod_i \exp\left\{-\frac{1}{2\sigma^2}\left(v_i^2 - 2v_i\left[a + \sigma \sum_\alpha \sum_{j \in \mathrm{N}(i)} h_j^\alpha \sum_k z_{jk}^\alpha \omega^\alpha(d(j,i),k)\right]\right)\right\}}{\frac{1}{Z} \exp\left\{\sum_{\alpha,j} b_\alpha h_j^\alpha\right\} \sum_{\mathbf{v}} \prod_i \exp\left\{-\frac{1}{2\sigma^2}\left(v_i^2 - 2v_i\left[a + \sigma \sum_\alpha \sum_{j \in \mathrm{N}(i)} h_j^\alpha \sum_k z_{jk}^\alpha \omega^\alpha(d(j,i),k)\right]\right)\right\}} \\
&= \prod_i \frac{\exp\left\{-\frac{1}{2\sigma^2}\left(v_i^2 - 2v_i\left[a + \sigma \sum_\alpha \sum_{j \in \mathrm{N}(i)} h_j^\alpha \sum_k z_{jk}^\alpha \omega^\alpha(d(j,i),k)\right]\right)\right\}}{\sum_{v_i} \exp\left\{-\frac{1}{2\sigma^2}\left(v_i^2 - 2v_i\left[a + \sigma \sum_\alpha \sum_{j \in \mathrm{N}(i)} h_j^\alpha \sum_k z_{jk}^\alpha \omega^\alpha(d(j,i),k)\right]\right)\right\}} \\
&= \prod_i p(v_i \mid \mathbf{h}, \mathbf{z}, \theta)\,,
\end{aligned}$$

where

$$p(v_i \mid \mathbf{h}, \mathbf{z}, \theta) = \mathcal{N}\left(v_i; a + \sigma \sum_\alpha \sum_{j \in \mathrm{N}(i)} h_j^\alpha \sum_k z_{jk}^\alpha \omega^\alpha(d(j,i),k), \sigma^2\right), \tag{B.6}$$

which is obtained by using the 'completing the square'-technique.

B.3 Gradients in training a rotation equivariant RBM (STEER-RBM)

The objective function used in the experiments was based on log-likelihood of data under the model:

$$\log p\big(\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(N)} \mid \mathbf{W}\big) = -N \log Z + \sum_{n=1}^N \sum_j \log g_j(\mathbf{v}^{(n)} | \mathbf{W}_j),$$

where

$$g_j(\mathbf{v}^{(n)} | \mathbf{W}_j) = K + \sum_{k=1}^K \exp\left\{[\mathbf{v}^{(n)}]^\top \mathbf{R}^{(k)} W_j(\cdot, 1)\right\}.$$

[3] If a hidden unit is not turned on (its state is zero), it is then not necessary to sample its rotation assignment variable.

The gradient of the objective function with respect to the canonical weights for feature j is thus

$$\frac{\partial \log p\big(\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(N)} \mid \mathbf{W}\big)}{\partial W_j(\cdot, 1)} = -N \frac{1}{Z} \frac{\partial Z}{\partial W_j(\cdot, 1)} + \sum_{n=1}^{N} \frac{1}{g_j(\mathbf{v}^{(n)}|\mathbf{W}_j)} \frac{\partial g_j(\mathbf{v}^{(n)}|\mathbf{W}_j)}{\partial W_j(\cdot, 1)}. \tag{B.7}$$

Using the fact that $\frac{\partial g_j(\mathbf{v}^{(n)}|\mathbf{W}_j)}{\partial W_j(\cdot,1)} = \sum_{k=1}^{K} \left[\mathbf{R}^{(k)}\right]^\top \mathbf{v}^{(n)} \exp\left\{\left[\mathbf{v}^{(n)}\right]^\top \mathbf{R}^{(k)} W^j_{\cdot 1}\right\}$, we obtain

$$\begin{aligned} \frac{1}{g_j(\mathbf{v}^{(n)}|\mathbf{W}_j)} \frac{\partial g_j(\mathbf{v}^{(n)}|\mathbf{W}_j)}{\partial W_j(\cdot, 1)} &= \frac{\sum_{k=1}^{K} \left[\mathbf{R}^{(k)}\right]^\top \mathbf{v}^{(n)} \exp\left\{\left[\mathbf{v}^{(n)}\right]^\top \mathbf{R}^{(k)} W^j_{\cdot 1}\right\}}{K + \sum_{k=1}^{K} \exp\left\{\left[\mathbf{v}^{(n)}\right]^\top \mathbf{R}^{(k)} W_j(\cdot, 1)\right\}} \\ &= \sum_{k=1}^{K} \left[\mathbf{R}^{(k)}\right]^\top \mathbf{v}^{(n)} p(z_j(k) = 1, h_j = 1 | \mathbf{v}^{(n)}, W). \end{aligned} \tag{B.8}$$

The gradient of the partition function Z is a bit more involved. Marginalizing out the visible units, we obtain

$$p(\mathbf{z}, \mathbf{h}|\mathbf{W}) = \frac{1}{Z} \prod_i \left(1 + \exp\left\{\sum_j h_j \left[\sum_{k=1}^{K} z_j(k) \mathbf{R}^{(k)}(i, \cdot)\right] W_j(\cdot, 1)\right\}\right) \tag{B.9}$$

$$= \frac{1}{Z} \prod_i f_i(\mathbf{h}, \mathbf{z}|\mathbf{W}), \tag{B.10}$$

where $f_i(\mathbf{h}, \mathbf{z}|\mathbf{W}) = \left(1 + \exp\left\{\sum_j h_j \left[\sum_{k=1}^{K} z_j(k) \mathbf{R}^{(k)}(i, \cdot)\right] W_j(\cdot, 1)\right\}\right)$. Therefore we find that

$$\begin{aligned} Z &= \sum_{\mathbf{h},\mathbf{z}} \prod_i f_i(\mathbf{h}, \mathbf{z}|\mathbf{W}) \\ &= \sum_{\mathbf{h},\mathbf{z}} \prod_i \left(1 + \exp\left\{\sum_j h_j \left[\sum_{k=1}^{K} z_j(k) \mathbf{R}^{(k)}(i, \cdot)\right] W_j(\cdot, 1)\right\}\right). \end{aligned} \tag{B.11}$$

Denoting the set visible unit indices as I, we obtain a representation for the gradient of Z:

$$\frac{\partial Z}{\partial W_j(\cdot, 1)} = \sum_{\mathbf{h},\mathbf{z}} \sum_i \left[\prod_{\ell \in I \setminus i} f_\ell(\mathbf{h}, \mathbf{z}|\mathbf{W})\right] \frac{\partial f_i(\mathbf{h}, \mathbf{z}|\mathbf{W})}{\partial W_j(\cdot, 1)} \tag{B.12}$$

$$= \sum_{\mathbf{h},\mathbf{z}} \sum_i \left[\prod_{i \in I} f_i(\mathbf{h}, \mathbf{z}|\mathbf{W})\right] \frac{1}{f_i(\mathbf{h}, \mathbf{z}|\mathbf{W})} \frac{\partial f_i(\mathbf{h}, \mathbf{z}|\mathbf{W})}{\partial W_j(\cdot, 1)} \tag{B.13}$$

$$= \sum_{\mathbf{h},\mathbf{z}} \prod_i f_i(\mathbf{h}, \mathbf{z}|\mathbf{W}) \sum_i \frac{1}{f_i(\mathbf{h}, \mathbf{z}|\mathbf{W})} \frac{\partial f_i(\mathbf{h}, \mathbf{z}|\mathbf{W})}{\partial W_j(\cdot, 1)}. \tag{B.14}$$

This can be then used to obtain the gradient of its logarithm:

$$\frac{\partial \log Z}{\partial W_j(\cdot,1)} = \frac{1}{Z}\frac{\partial Z}{\partial W_j(\cdot,1)} = \sum_{\mathbf{h},\mathbf{z}} \frac{1}{Z}\prod_i f_i(\mathbf{h},\mathbf{z}|\mathbf{W}) \sum_i \frac{1}{f_i(\mathbf{h},\mathbf{z}|\mathbf{W})}\frac{\partial f_i(\mathbf{h},\mathbf{z}|\mathbf{W})}{\partial W_j(\cdot,1)}. \tag{B.15}$$

Because $\frac{1}{Z}\prod_i f_i(\mathbf{h},\mathbf{z}|\mathbf{W}) = p(\mathbf{z},\mathbf{h}|\mathbf{W})$, we obtain

$$\frac{\partial \log Z}{\partial W_j(\cdot,1)} = \sum_{\mathbf{h},\mathbf{z}} p(\mathbf{h},\mathbf{z}|\mathbf{W}) \sum_i \frac{1}{f_i(\mathbf{h},\mathbf{z}|\mathbf{W})}\frac{\partial f_i(\mathbf{h},\mathbf{z}|\mathbf{W})}{\partial W_j(\cdot,1)}, \tag{B.16}$$

where

$$\frac{\partial f_i(\mathbf{h},\mathbf{z}|\mathbf{W})}{\partial W_j(\cdot,1)} = h_j\left[\sum_{k=1}^{K} z_j(k)\mathbf{R}^{(k)}(i,\cdot)\right]^\top \exp\left\{\sum_j h_j\left[\sum_{k=1}^{K} z_j(k)\mathbf{R}^{(k)}(i,\cdot)\right] W_j(\cdot,1)\right\}. \tag{B.17}$$

By plugging in terms into (B.15) and simplifying, we obtain

$$\frac{\partial \log Z}{\partial W_j(\cdot,1)} = \sum_{\mathbf{h},\mathbf{z}} p(\mathbf{h},\mathbf{z}|\mathbf{W}) \sum_i h_j\left[\sum_{k=1}^{K} z_j(k)\mathbf{R}^{(k)}(i,\cdot)\right]^\top \text{logistic}\left(\sum_j h_j\left[\sum_{k=1}^{K} z_j(k)\mathbf{R}^{(k)}(i,\cdot)\right] W_j(\cdot,1)\right). \tag{B.18}$$

Using an auxiliary variable vector $\mathbf{c} = [1 \ \ldots \ K]^\top$ (which picks the rotation assignment number of unit j, when inner-producted with $\mathbf{z}_j$) we arrive at an alternative form

$$\frac{\partial \log Z}{\partial W_j(\cdot,1)} = \sum_{\mathbf{h},\mathbf{z}} p(\mathbf{h},\mathbf{z}|\mathbf{W}) \sum_i h_j\left[\mathbf{R}^{(\mathbf{c}^\top\mathbf{z}_j)}(i,\cdot)\right]^\top \text{logistic}\left(\sum_j h_j \mathbf{R}^{(\mathbf{c}^\top\mathbf{z}_j)}(i,\cdot) W_j(\cdot,1)\right). \tag{B.19}$$

Plugging this and (B.8) in (B.7) and simplifying, we finally obtain that

$$\begin{aligned}
\frac{\partial \log p\big(\mathbf{v}^{(1)},\ldots,\mathbf{v}^{(N)} \mid \mathbf{W}\big)}{\partial W_j(\cdot,1)} &= \sum_{n=1}^{N}\left(\frac{\sum_{k=1}^{K}\left[\mathbf{R}^{(k)}\right]^\top \mathbf{v}^{(n)} \exp\left\{\left[\mathbf{v}^{(n)}\right]^\top \mathbf{R}^{(k)} W_{\cdot 1}^{j}\right\}}{K + \sum_{k=1}^{K}\exp\left\{\left[\mathbf{v}^{(n)}\right]^\top \mathbf{R}^{(k)} W_j(\cdot,1)\right\}}\right) \\
&- N\sum_{\mathbf{h},\mathbf{z}} p(\mathbf{h},\mathbf{z}|\mathbf{W}) \sum_i h_j\left[\sum_{k=1}^{K} z_j(k)\mathbf{R}^{(k)}(i,\cdot)\right]^\top \text{logistic}\left(\sum_j h_j\left[\sum_{k=1}^{K} z_j(k)\mathbf{R}^{(k)}(i,\cdot)\right] W_j(\cdot,1)\right) \\
&= \sum_{n=1}^{N}\sum_{k=1}^{K}\left[\mathbf{R}^{(k)}\right]^\top \mathbf{v}^{(n)} p(z_j(k)=1, h_j=1|\mathbf{v}^{(n)}, W) \\
&- N\sum_{\mathbf{h},\mathbf{z}} p(\mathbf{h},\mathbf{z}|\mathbf{W}) \sum_i h_j\left[\mathbf{R}^{(\mathbf{c}^\top\mathbf{z}_j)}(i,\cdot)\right]^\top \text{logistic}\left(\sum_j h_j \mathbf{R}^{(\mathbf{c}^\top\mathbf{z}_j)}(i,\cdot) W_j(\cdot,1)\right).
\end{aligned} \tag{B.20}$$

B.4 Gradients in training a STEER-DBN

B.4.1 Partial derivatives related to training the first level STEER-RBM

Assuming a convolutional STEER-RBM as defined in (3.3) (and constant σ) the partial derivatives are:

$$\frac{\partial \mathrm{E}(\mathbf{v},\mathbf{h},\mathbf{z}\mid\theta)}{\partial a} = -\frac{1}{\sigma^2}\sum_i v_i \qquad \frac{\partial \mathrm{E}(\mathbf{v},\mathbf{h},\mathbf{z}\mid\theta)}{\partial b_\alpha} = -\sum_j h_j^\alpha \tag{B.21}$$

$$\frac{\partial \mathrm{E}(\mathbf{v},\mathbf{h},\mathbf{z}\mid\theta)}{\partial \omega^\alpha(\beta,1)} = -\frac{1}{\sigma}\sum_j h_j^\alpha \sum_{k=1}^K z_{jk}^\alpha \sum_{\ell\in \mathrm{N}_{\alpha j}} v_\ell \mathbf{R}^{(k)}(d(j,\ell),\beta) \tag{B.22}$$

$$= -\frac{1}{\sigma}\sum_j h_j^\alpha \sum_{k=1}^K z_{jk}^\alpha \big[\mathbf{R}^{(k)}(\cdot,\beta)\big]^\top v_{\mathrm{N}_{\alpha j}} \tag{B.23}$$

$$= -\frac{1}{\sigma}\sum_j h_j^\alpha \Big[\mathbf{R}^{(\mathbf{c}^\top \mathbf{z}_j^\alpha)}(\cdot,\beta)\Big]^\top v_{\mathrm{N}_{\alpha j}} \tag{B.24}$$

$$\frac{\partial p\Big(\mathbf{h}_j^{\alpha(n)}=1\mid \mathbf{v}^{(n)},\theta\Big)}{\partial b_\alpha} = \frac{K\sum_{k=1}^K \exp\Big\{b_\alpha + \frac{1}{\sigma}\sum_{\ell\in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\Big\}}{\Big[K+\sum_{k=1}^K \exp\Big\{b_\alpha + \frac{1}{\sigma}\sum_{\ell\in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\Big\}\Big]^2} \tag{B.25}$$

$$\frac{\partial p\Big(\mathbf{h}_j^{\alpha(n)}=1\mid \mathbf{v}^{(n)},\theta\Big)}{\partial \omega^\alpha(\beta,1)} = \frac{K\sum_{k=1}^K \sum_{\ell\in \mathrm{N}_{\alpha j}} \frac{1}{\sigma} v_\ell \mathbf{R}^{(k)}(d(j,\ell),\beta)\exp\Big\{b_\alpha + \frac{1}{\sigma}\sum_{\ell\in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\Big\}}{\Big[K+\sum_{k=1}^K \exp\Big\{b_\alpha + \frac{1}{\sigma}\sum_{\ell\in \mathrm{N}_{\alpha j}} v_\ell \omega^\alpha(d(j,\ell),k)\Big\}\Big]^2}. \tag{B.26}$$

B.4.2 Partial derivatives related to training the higher level STEER-RBMs

Consider a STEER-RBM at level $\ell > 1$ in a deep structure, and considering it as a stand-alone STEER-RBM. The energy function for the stand-alone model can be then written as

$$\mathrm{E}\big(\mathbf{h}^{\ell-1},\mathbf{z}^{\ell-1},\mathbf{h}^{\ell},\mathbf{z}^{\ell}\mid\theta^{\ell}\big) = -\sum_\delta b_\delta^{\ell-1}\sum_i h_i^{\ell-1_\delta} - \sum_\alpha b_\alpha^\ell \sum_j h_j^{\ell_\alpha} - \sum_\alpha \sum_j h_j^{\ell_\alpha} \sum_\delta \sum_{i\in \mathrm{N}_{\alpha j}} h_i^{\ell-1_\delta} \omega_{\ell-1_\delta}^{\ell_\alpha}(d(j,i),\mathbf{c}^\top \mathbf{z}_i^{\ell-1_\delta},\mathbf{c}^\top \mathbf{z}_j^{\ell_\alpha}), \tag{B.27}$$

where as before, the auxiliary variable vector $\mathbf{c} = [1 \ \dots \ K]^\top$. The partial derivatives of this energy function with respect to model parameters are then the following:

$$\frac{\partial \mathrm{E}\big(\mathbf{h}^{\ell-1}, \mathbf{z}^{\ell-1}, \mathbf{h}^{\ell}, \mathbf{z}^{\ell} \mid \theta^{\ell}\big)}{\partial b_{\delta}^{\ell-1}} = -\sum_{i} h_i^{\ell-1_\delta} \qquad \frac{\partial \mathrm{E}\big(\mathbf{h}^{\ell-1}, \mathbf{z}^{\ell-1}, \mathbf{h}^{\ell}, \mathbf{z}^{\ell} \mid \theta^{\ell}\big)}{\partial b_{\alpha}^{\ell}} = -\sum_{j} h_j^{\ell_\alpha} \tag{B.28}$$

$$\frac{\partial \mathrm{E}\big(\mathbf{h}^{\ell-1}, \mathbf{z}^{\ell-1}, \mathbf{h}^{\ell}, \mathbf{z}^{\ell} \mid \theta^{\ell}\big)}{\partial \omega_{\ell-1_\delta}^{\ell_\alpha}(\beta, \gamma, 1)} =$$

$$-\sum_{j} h_j^{\ell_\alpha} \sum_{k=1}^{K} z_{jk}^{\ell_\alpha} \sum_{i \in \mathrm{N}_{\alpha j}} h_i^{\ell-1_\delta} \mathbf{R}^{(k)}(d(j,i), \beta) \sum_{m=1}^{K} z_{im}^{\ell-1_\delta} \mathbf{S}^{(k)}(\gamma, m) \tag{B.29}$$

$$= -\sum_{j} h_j^{\ell_\alpha} \sum_{i \in \mathrm{N}_{\alpha j}} h_i^{\ell-1_\delta} \mathbf{R}^{(\mathbf{c}^\top \mathbf{z}_j^{\ell_\alpha})}(d(j,i), \beta) \mathbf{S}^{(\mathbf{c}^\top \mathbf{z}_j^{\ell_\alpha})}(\gamma, \mathbf{c}^\top \mathbf{z}_i^{\ell-1_\delta}). \tag{B.30}$$

Let us now consider the partial derivatives $\frac{\partial p\left(h_j^{\ell_\alpha}=1 \mid \mathbf{z}_j^{\ell_\alpha}, \mathbf{h}^{\ell-1}, \mathbf{z}^{\ell-1}, \theta^{\ell}\right)}{\partial \theta^{\ell}}$.

$$p\big(h_j^{\ell_\alpha} = 1 \mid \mathbf{z}_j^{\ell_\alpha}, \mathbf{h}^{\ell-1}, \mathbf{z}^{\ell-1}, \theta^{\ell}\big) = \frac{\sum_{k=1}^{K} \exp\{A(k)\}}{K + \sum_{k=1}^{K} \exp\{A(k)\}}, \tag{B.31}$$

where

$$A(k) = b_{\alpha}^{\ell} + \sum_{\delta} \sum_{i \in \mathrm{N}_{\alpha j}} h_i^{\ell-1_\delta} \omega_{\ell-1_\delta}^{\ell_\alpha}(d(j,i), \mathbf{c}^\top \mathbf{z}_i^{\ell-1_\delta}, k).$$

Therefore,

$$\frac{\partial p\big(h_j^{\ell_\alpha} = 1 \mid \mathbf{z}_j^{\ell_\alpha}, \mathbf{h}^{\ell-1}, \mathbf{z}^{\ell-1}, \theta^{\ell}\big)}{\partial \phi} = \frac{K}{\Big[K + \sum_{k=1}^{K} \exp\{A(k)\}\Big]^2} \sum_{k=1}^{K} \exp\{A(k)\} \frac{\partial A(k)}{\partial \phi}. \tag{B.32}$$

In the case $\phi = b_{\alpha}^{\ell}$, $\frac{\partial A(k)}{\partial \phi} = 1$. The case $\phi = \omega_{\ell-1_\delta}^{\ell_\alpha}(\beta, \gamma, 1)$ is more complicated:

$$\frac{\partial A(k)}{\partial \omega_{\ell-1_\delta}^{\ell_\alpha}(\beta, \gamma, 1)} = \sum_{i \in \mathrm{N}_{\alpha j}} h_i^{\ell-1_\delta} \mathbf{R}^{(k)}(d(j,i), \beta) \mathbf{S}^{(k)}(\gamma, \mathbf{c}^\top \mathbf{z}_i^{\ell-1_\delta}). \tag{B.33}$$

Appendix C

Texture Modelling

In the following section we develop and assess methods for handling boundaries, an issue empirically found to be of significant practical importance. We also study the effect of using hidden unit augmentations and block Gibbs sampling versus integrating them out and using hybrid Monte Carlo (HMC) in learning and texture synthesis tasks. In both of these studies we focus on the GB-RBM with tiled-convolutional feature sharing, which is the building block of the multiple texture model developed in Chapter 4. Section C.2 gives partial derivatives for gradient-based learning of texture models in Chapter 4.

C.1 Practical Modelling Considerations

There are many practical considerations involved with training and inference with the texture models of Chapter 4. This section considers two such issues, namely the use of hidden units in training and inference; and handling image boundaries.

Although the models considered in experiments in Sec. 4.3 have hidden unit augmentations available, and are amenable to block Gibbs sampling, the joint conditional distribution of visible units given hidden units factorizes over sites only with the GB-RBM model. For the other models the distribution to sample from is a multivariate Gaussian distribution, with potentially full precision matrix over full-image dimensions, and for large images the sampling can be complicated in practice.

To study the effect of using hidden unit augmentations to generative performance, the following subsection C.1.1 analyzes the texture synthesis quality of a GB-RBM with tiled-convolutional feature sharing (Tm) when hidden units

are either used or integrated out in training and/or testing. Based on these results, subsection C.1.2 analyzes several boundary handling methods under the best augmentation setup. The boundary handling methods include the approach considered above, and three other ones.

We then discuss how to further extend the model to deal with colour images, using receptive fields at various scales, and making the feature sharing data-adaptive, for making the model applicable to very generic scenarios.

C.1.1 The use of latent variables

Here we assess the effect of using latent variables to synthesis performance of a Tm-model by analyzing unconstrained and constrained texture synthesis performance under two Brodatz textures (D21,D77) when latent variables are used or integrated out in training and/or in testing. Similar to the experiments in Sec. 4.3 and 4.4, the train/test data are top/bottom parts of the respective texture images.

In our experiments the parameter update formulas in training were set to be equivalent for all of the cases. That is also the case for the values of the hyperparameters. The only thing that was different was the way negative particles or model samples were drawn/updated: when latent variables were used, negative particles or model samples were drawn/updated by using block-Gibbs sampling (BG). In the case when the hidden units were integrated out, hybrid Monte Carlo (HMC) was used instead.

C.1.1.1 Training

The models were trained using FPCD, similar to the techniques in the previous section. A batch size of 128 samples of size 71×71 was used, of which for negative particles boundaries of 7 units wide were clamped to zero. The training algorithm was run for 50,000 iterations, using a learning rate 0.0005 for weights, and 0.032 for visible biases. The learning rates were annealed after 5,000 iterations, such that they assumed $\frac{1}{t}$-type annealing, adjusted so that the learning rates were 0.1 of the initial learning rate at the end of the training. The HMC-based negative particle updates used 30 Leapfrog-updates, with step sizes adjusted for 90 percent acceptance rate.

C.1.1.2 Unconstrained texture synthesis

Samples were collected by storing the states of 128 independently and randomly initialized chains after 100,000 iterations of Markov chain Monte Carlo for each analysis case. Figure C.3 shows example model samples, and raw test data patches of similar size. The quality of the model samples were analyzed quantitatively by computing TSS-scores of the samples to the test portions of the corresponding textures, at various matching window sizes. These results are summarized in Table C.1, and in Figures C.1 (D21), and C.2 (D77). Assuming a trained model, the results are largely insensitive to whether block-Gibbs sampling (BG) or HMC is used, provided the model was not exhibiting strong failure indications or artifacts when compared to the ground truth as with the D77 using HMC in training. The D77 synthesis results of models trained with HMC based negative particle updating are clearly weaker than those of based on block-Gibbs sampling, as is also clear from Figure C.3. Overall there seems to be slightly more robust performance associated with the models trained with using hidden units and block-Gibbs sampling. However, as our results in Chapter 4 demonstrate, excellent performance (under several textures which include the considered textures here) can be obtained by using HMC to draw negative particles in model training. In this appendix we have used considerably less time in choosing the hyperparameters, and there are several differences to the setup in Chapter 4. The best performance is, however, clearly achieved with an approach modified from the BG sampling denoted as BG-. In this approach the inference result is set as the conditional mean of visible units given the last state of hidden units as opposed to a sample from the joint conditional distribution, and yields clear improvements over the BG approach on both textures.

C.1.1.3 Constrained texture synthesis

Inpainting performance was assessed on 20 cases per texture, each consisting of a 79×79 unit image, having 11-unit ground truth borders, and a 57×57 inpainting hole in the center. Each of the cases had a corresponding ground truth texture patch within the test portion of the corresponding Brodatz-texture, and their positions were chosen randomly.

Inpainting was performed for each case by running the Markov chain Monte Carlo algorithms with 5 chains started from the inpainting frames each associated

Brodatz Texture D21 Synthesis (TSS):

	Sampling Approach (Learning, Inference)					
Window Size	HMC,HMC	HMC,BG	HMC,BG-	BG,HMC	BG,BG	BG,BG-
19×19	0.9208 ± 0.0243	0.9175 ± 0.0310	0.9465 ± 0.0302	0.9025 ± 0.0361	0.9109 ± 0.0300	0.9394 ± 0.0299
25×25	0.9138 ± 0.0200	0.9039 ± 0.0548	0.9315 ± 0.0574	0.8930 ± 0.0418	0.8968 ± 0.0383	0.9246 ± 0.0393
31×31	0.9008 ± 0.0326	0.8858 ± 0.0723	0.9142 ± 0.0735	0.8804 ± 0.0463	0.8905 ± 0.0351	0.9190 ± 0.0360
37×37	0.8926 ± 0.0484	0.8760 ± 0.0774	0.9046 ± 0.0790	0.8738 ± 0.0447	0.8797 ± 0.0446	0.9078 ± 0.0461
43×43	0.8753 ± 0.0853	0.8733 ± 0.0616	0.9009 ± 0.0635	0.8636 ± 0.0575	0.8741 ± 0.0488	0.9020 ± 0.0504
49×49	0.8759 ± 0.0544	0.8588 ± 0.0909	0.8867 ± 0.0933	0.8561 ± 0.0592	0.8685 ± 0.0460	0.8965 ± 0.0473
55×55	0.8653 ± 0.0814	0.8531 ± 0.0929	0.8807 ± 0.0959	0.8525 ± 0.0560	0.8625 ± 0.0536	0.8904 ± 0.0556
61×61	0.8572 ± 0.0908	0.8449 ± 0.0967	0.8724 ± 0.1000	0.8467 ± 0.0659	0.8577 ± 0.0535	0.8852 ± 0.0553
67×67	0.8519 ± 0.0884	0.8360 ± 0.1049	0.8634 ± 0.1080	0.8419 ± 0.0638	0.8519 ± 0.0583	0.8795 ± 0.0599
73×73	0.8455 ± 0.0904	0.8290 ± 0.1060	0.8561 ± 0.1090	0.8367 ± 0.0659	0.8474 ± 0.0569	0.8749 ± 0.0585

Brodatz Texture D77 Synthesis (TSS):

	Sampling Approach (Learning, Inference)					
Window Size	HMC,HMC	HMC,BG	HMC,BG-	BG,HMC	BG,BG	BG,BG-
19×19	0.7674 ± 0.1044	0.7202 ± 0.1521	0.7672 ± 0.1312	0.8196 ± 0.0156	0.8224 ± 0.0134	0.8560 ± 0.0133
25×25	0.7459 ± 0.1021	0.7062 ± 0.1417	0.7496 ± 0.1299	0.7994 ± 0.0160	0.7997 ± 0.0165	0.8328 ± 0.0164
31×31	0.7414 ± 0.0770	0.6969 ± 0.1284	0.7380 ± 0.1187	0.7841 ± 0.0152	0.7830 ± 0.0202	0.8163 ± 0.0208
37×37	0.7217 ± 0.0769	0.6644 ± 0.1353	0.7046 ± 0.1283	0.7726 ± 0.0160	0.7724 ± 0.0237	0.8054 ± 0.0252
43×43	0.7111 ± 0.0683	0.6495 ± 0.1272	0.6891 ± 0.1200	0.7621 ± 0.0138	0.7617 ± 0.0254	0.7947 ± 0.0258
49×49	0.6973 ± 0.0669	0.6429 ± 0.1274	0.6805 ± 0.1230	0.7536 ± 0.0146	0.7532 ± 0.0302	0.7857 ± 0.0310
55×55	0.6803 ± 0.0650	0.6243 ± 0.1308	0.6607 ± 0.1269	0.7453 ± 0.0167	0.7440 ± 0.0272	0.7767 ± 0.0276
61×61	0.6682 ± 0.0602	0.6122 ± 0.1261	0.6482 ± 0.1229	0.7347 ± 0.0196	0.7349 ± 0.0317	0.7669 ± 0.0328
67×67	0.6547 ± 0.0612	0.5997 ± 0.1248	0.6337 ± 0.1236	0.7255 ± 0.0209	0.7245 ± 0.0331	0.7559 ± 0.0342
73×73	0.6350 ± 0.0644	0.5818 ± 0.1236	0.6134 ± 0.1236	0.7130 ± 0.0224	0.7118 ± 0.0330	0.7417 ± 0.0344

Table C.1: Sample means and standard deviations of the texture synthesis TSS-scores for D21 (top) and D77 (bottom), under different approaches which use different (training,testing) sampling method combinations (columns) and using different window sizes in the score computation (rows). HMC denotes hybrid Monte Carlo, BG denotes block-Gibbs, and BG- denotes block-Gibbs with last state of visible units in the sampling set as the mean of the joint conditional distribution of visible units given hidden units.

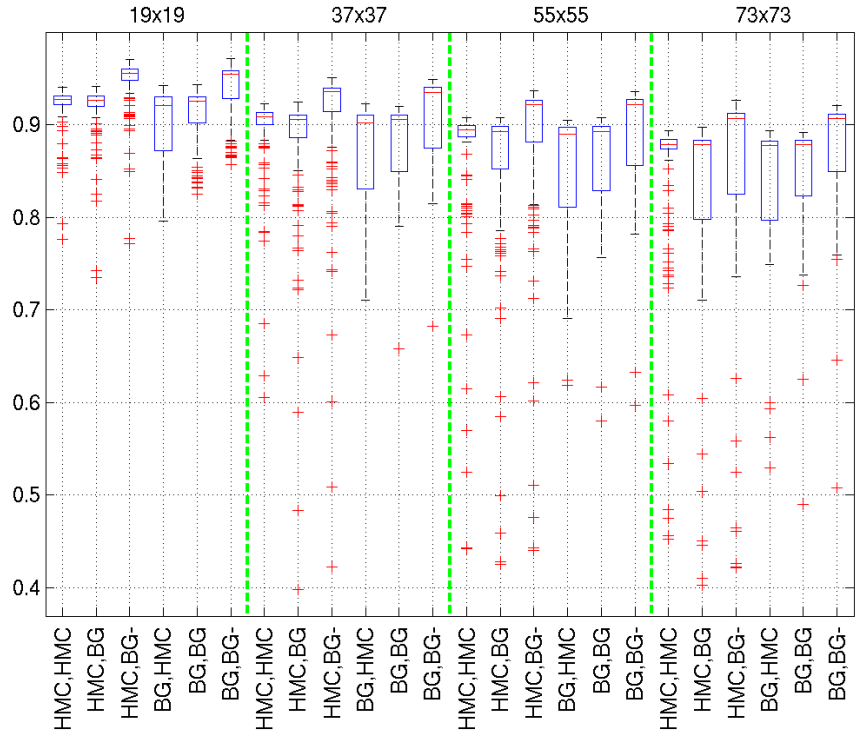


Figure C.1: Effect of (train,test) sampling method combination to synthesis quality: D21 Brodatz texture synthesis quality assessment for the methods measured as texture similarity score between model samples and the testing half of the texture, using different window sizes in the score computation (horizontal blocks). Boxes indicate the upper and lower quartiles as well as the median (red bar) of the TSS distributions; whiskers show extent of the rest of the data; red crosses denote outliers. HMC denotes hybrid Monte Carlo, BG denotes block-Gibbs, and BG- denotes block-Gibbs with last state of visible units in the sampling set as the mean of the joint conditional distribution of visible units given hidden units.

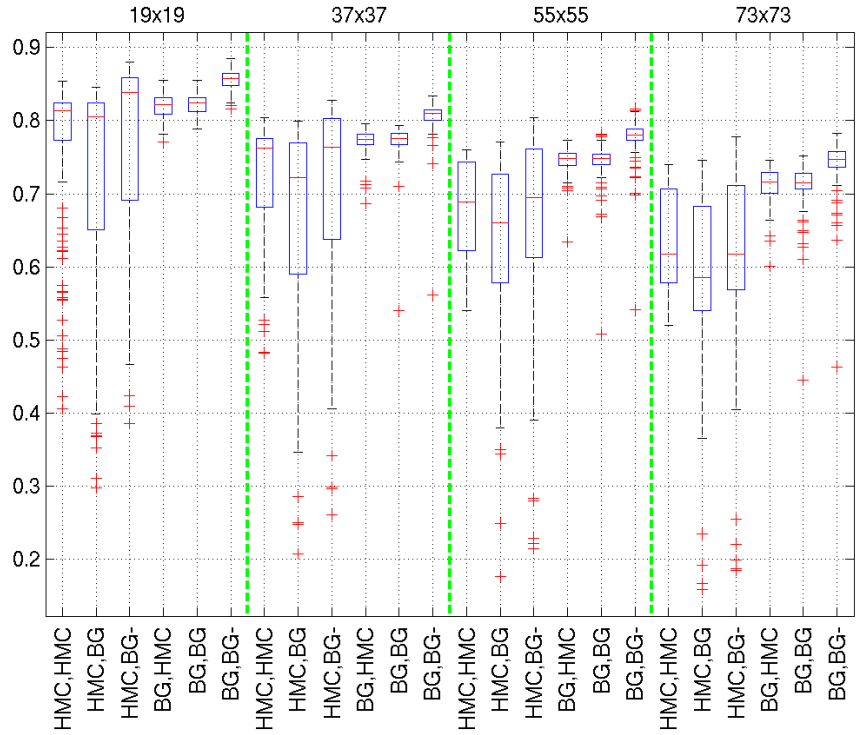


Figure C.2: Effect of (train,test) sampling method combination to synthesis quality: D77 Brodatz texture synthesis quality assessment for the methods measured as texture similarity score between model samples and the testing half of the texture, using different window sizes in the score computation (horizontal blocks). Boxes indicate the upper and lower quartiles as well as the median (red bar) of the TSS distributions; whiskers show extent of the rest of the data; red crosses denote outliers. HMC denotes hybrid Monte Carlo, BG denotes block-Gibbs, and BG- denotes block-Gibbs with last state of visible units in the sampling set as the mean of the joint conditional distribution of visible units given hidden units.

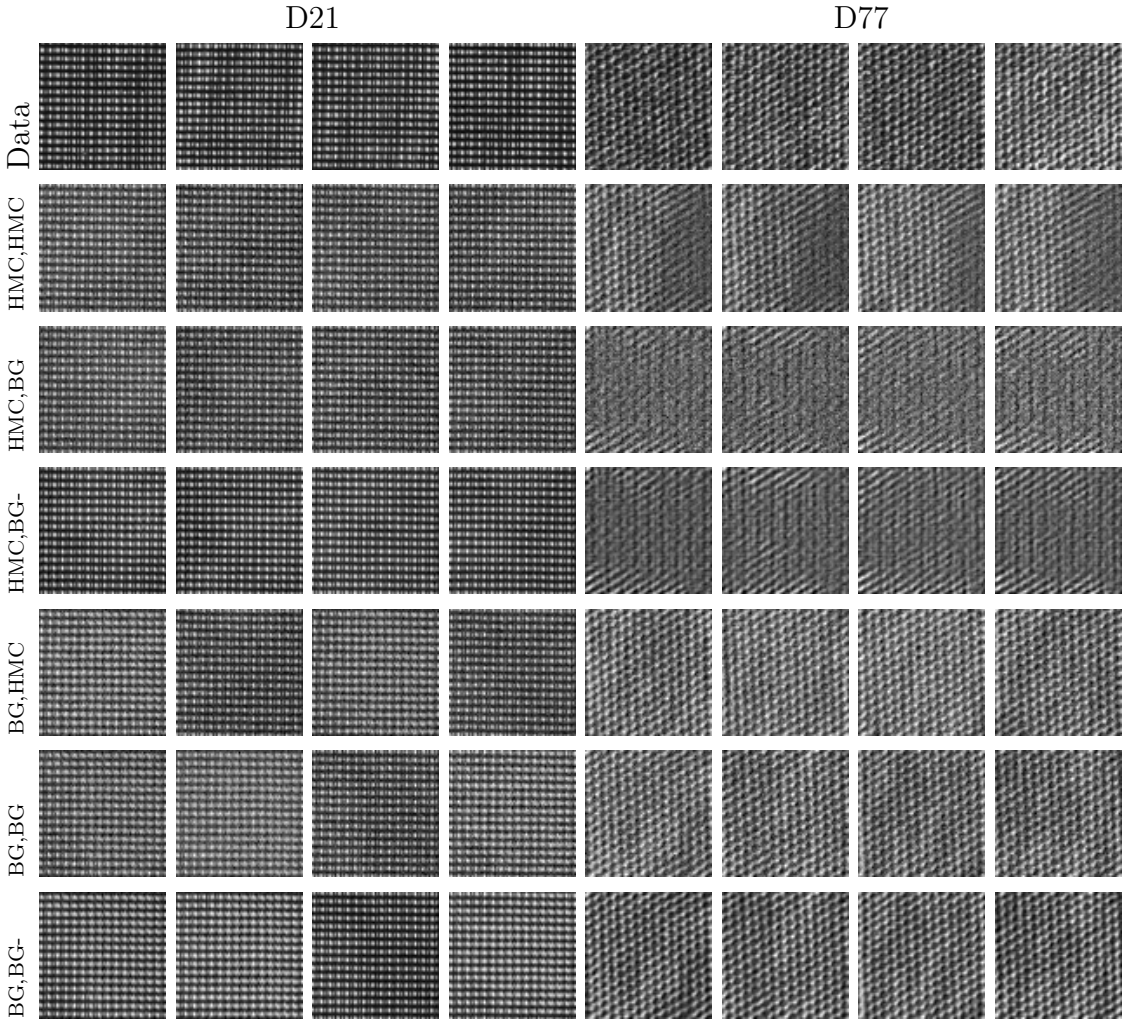


Figure C.3: Example data patches (top row), and samples under different approaches which use different (training,testing) sampling method combinations (other rows), with each case scaled independently to cover the full intensity range. HMC denotes hybrid Monte Carlo, BG denotes block-Gibbs, and BG- denotes block-Gibbs with last state of visible units in the sampling set as the mean of the joint conditional distribution of visible units given hidden units.

with different random number generator seed for 10,000 iterations, and storing the last states as result images. Figure C.5 shows examples of these.

To obtain quantitative results, for each sample the normalized cross-correlation (NCC) and mean structural similarity index (MSSIM) were computed between inpainted region and the corresponding ground truth texture region, and the texture similarity score between the inpainted region and the test portion of the corresponding texture. These results are summarized in Figure C.4 and in Table C.2. Similar to unconstrained synthesis, assuming a trained model the choice of the sampling method does not matter. This also should be the case for valid samplers assuming they have burnt in. Also similar to unconstrained synthesis, skipping the addition of i.i.d. zero-mean white Gaussian noise in the last visible unit update step in block-Gibbs sampling yields better results, and the BG- approach gives the best performance in general. Using the conditional mean as the estimator can be seen as applying *Rao-Blackwellization* [Blackwell, 1947].

C.1.1.4 Discussion

In terms of texture synthesis performance the issue of the use of hidden unit augmentations and using block-Gibbs versus integrating them out and using HMC did not largely matter. We note that there are many caveats in the study and therefore cannot draw strong conclusions for generic situations. It is however expected that using the Gibbs sampling is simpler to adopt, which is also the typical scenario for the models considered.

The results, however, suggested clear benefits of using the conditional mean of visible units given hidden units as opposed to a sample from the joint distribution of visible units given hidden units as the inference result in both unconstrained and constrained synthesis. The latter can be obtained from the former by adding i.i.d. zero-mean white Gaussian noise with standard deviation σ. While with the former the inference is not based on a sample from the model, the results suggest practical benefits of the approach for the tasks considered. Using the conditional mean as the estimator can be seen as applying *Rao-Blackwellization*. These results are in line with those obtained with the Tm and the Multi-Tm (256) in Sec. 4.4.

In the following subsection where different boundary handling methods are compared, both learning and inference use hidden unit augmentation and block-Gibbs sampling, and the inferences are based on conditional means of visible

Brodatz Texture D21 Inpainting:

	Sampling Approach (Learning, Inference)					
Metric	HMC,HMC	HMC,BG	HMC,BG-	BG,HMC	BG,BG	BG,BG-
NCC	0.8854 ± 0.0129	0.8865 ± 0.0131	0.9147 ± 0.0131	0.8864 ± 0.0109	0.8873 ± 0.0117	0.9155 ± 0.0117
MSSIM	0.8773 ± 0.0164	0.8784 ± 0.0166	0.9072 ± 0.0167	0.8784 ± 0.0140	0.8795 ± 0.0151	0.9083 ± 0.0153
TSS	0.8960 ± 0.0047	0.8966 ± 0.0047	0.9252 ± 0.0046	0.8953 ± 0.0057	0.8962 ± 0.0052	0.9246 ± 0.0055

Brodatz Texture D77 Inpainting:

	Sampling Approach (Learning, Inference)					
Metric	HMC,HMC	HMC,BG	HMC,BG-	BG,HMC	BG,BG	BG,BG-
NCC	0.7332 ± 0.0213	0.7306 ± 0.0195	0.7625 ± 0.0198	0.7339 ± 0.0195	0.7330 ± 0.0193	0.7646 ± 0.0194
MSSIM	0.7304 ± 0.0245	0.7281 ± 0.0222	0.7559 ± 0.0238	0.7339 ± 0.0189	0.7329 ± 0.0175	0.7605 ± 0.0184
TSS	0.7543 ± 0.0139	0.7530 ± 0.0134	0.7858 ± 0.0132	0.7556 ± 0.0100	0.7562 ± 0.0099	0.7886 ± 0.0103

Table C.2: Sample means and standard deviations of the texture inpainting scores for D21 (top) and D77 (bottom), under different approaches which use different (training,testing) sampling method combinations (columns), and under different performance metrics (rows). HMC denotes hybrid Monte Carlo, BG denotes block-Gibbs, and BG- denotes block-Gibbs with last state of visible units in the sampling set as the mean of the joint conditional distribution of visible units given hidden units.

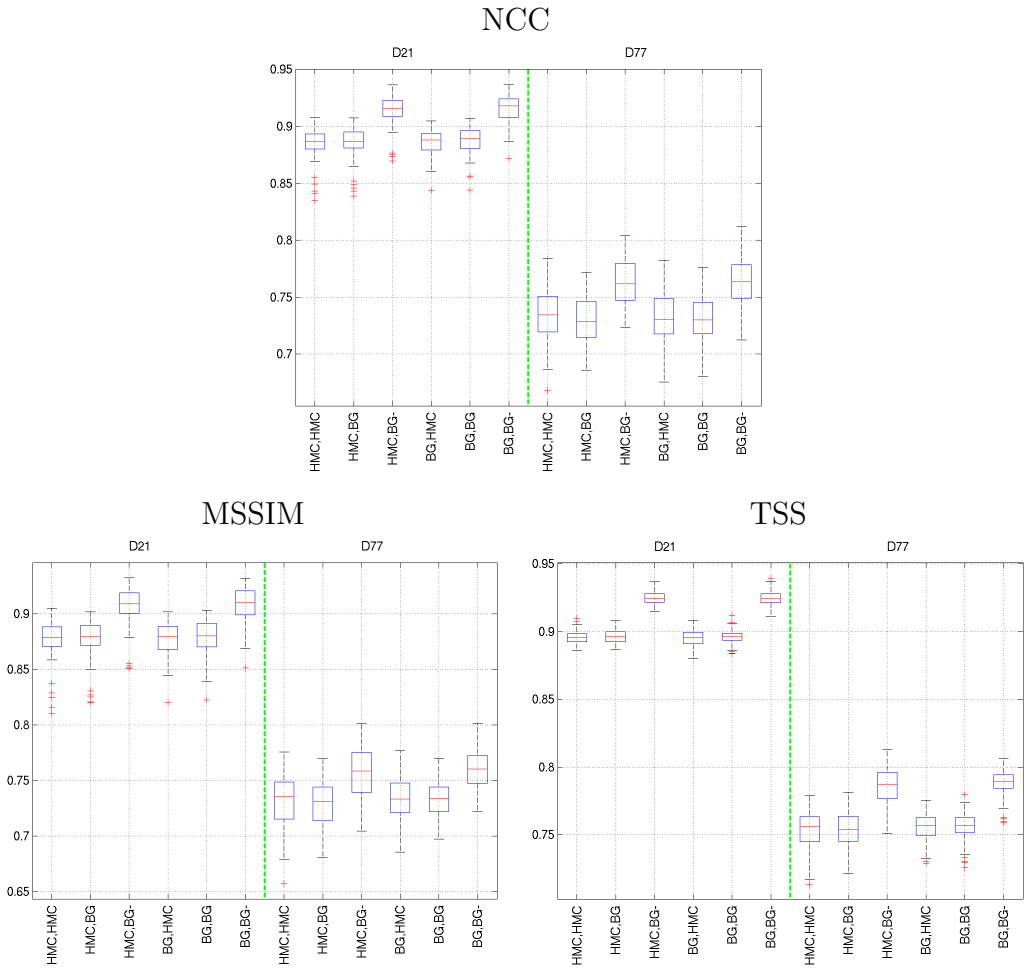


Figure C.4: Quality assessment for the different approaches which use different (training,testing) sampling method combinations, based on NCC/MSSIM/TSS between inpainted area and corresponding Brodatz texture. Boxes indicate the upper and lower quartiles as well as the median (red bar) of the TSS/NCC distributions; whiskers show extent of the rest of the data; red crosses denote outliers. HMC denotes hybrid Monte Carlo, BG denotes block-Gibbs, and BG- denotes block-Gibbs with last state of visible units in the sampling set as the mean of the joint conditional distribution of visible units given hidden units.

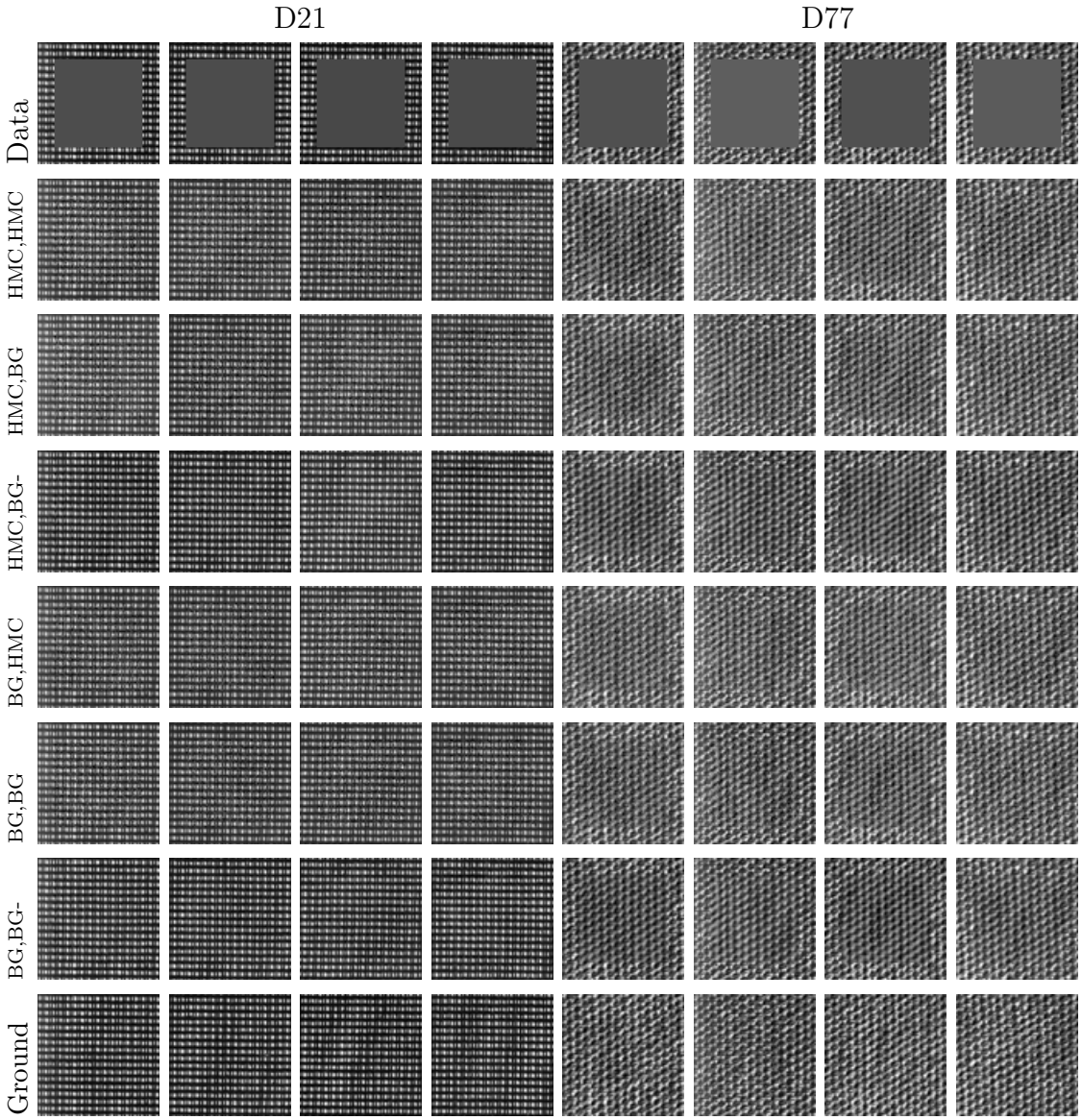


Figure C.5: Example inpainting frames (top row), ground truth (bottom), and approach-specific samples (other rows; sampling method used in training, sampling method used in testing), with each case scaled independently to cover the full intensity range. HMC denotes hybrid Monte Carlo, BG denotes block-Gibbs, and BG- denotes block-Gibbs with last state of visible units in the sampling set as the mean of the joint conditional distribution of visible units given hidden units.

units given the last states of hidden units in the block-Gibbs sampling scheme, adopting the most effective setup of the study in this subsection.

C.1.2 Boundary handling

In this subsection we discuss and develop methods for handling boundaries when using tiled-convolutional feature sharing, and evaluate the effectiveness of them in texture synthesis. In the experiments we use the Gaussian-Bernoulli RBM (Tm) as the image model, but the techniques are applicable to other models as well.

C.1.2.1 Methods for handling boundaries

We consider four different methods for handling boundaries, one of which (negBordersZero) has been used in the previous experiments in this chapter with the Tm-models. The approaches use the default visible and hidden unit grid spacings associated with the models, and do not use a wrap-around connectivity. The way they deal with boundary units are the following:

negBordersZero (NBoZ) Border visible units (the width of the square filter/receptive field minus one units assuming a stride of one) are clamped to zeros for the negative phase data, and when sampling from the model.

bordersZero (BoZ) This is similar to the above, with the exception that during training the positive phase data is also masked to zeros at the borders.

specialBiases (SBi) The approach assumes the same connectivity structure as the above. However, the parameterisation is different (extended): Each of the feature planes of the hidden units are associated with a (global) bias as in the above. However, these are scaled (multiplied by a scalar) and shifted (a scalar is added) at different sites. The scalars are shared across feature planes, and also assume a sharing based on the spatial pattern of weight connectivity to it. The number of these scaling and shifting variables is the number of unique weight connectivity patterns, and is independent of the number of features used per site. In practice the number is also independent of the lattice size due to the spatial sharing, and dependent only on the receptive field size, and the stride used in tiling. In the experiments here (in which the receptive field size was 8×8, and the stride was 1) the number

of both shifting and scaling variables was 62, a relatively small amount of variables when compared to the total amount of parameters in the model (the model also had 256 hidden unit biases (8 sets of 32 biases, one set per tiling) and 16,384 weights (8 sets of 32 weights of size 8×8)). This was also the amount of extra parameters when compared to the other boundary handling methods in this study.

latentBorders (LBo) Border visible units for the positive phase data are considered to be missing. In our experiments with this approach each of the hidden units were associated with the same number of visible units, and at the borders there were visible units which did not receive inputs with all the possible connectivity weights, and all of those units were considered to be latent/missing. There is no special treatment for the negative phase.

C.1.2.2 Learning

The models were trained using FPCD, using the techniques of previous subsection assuming the use of hidden units and block-Gibbs sampling to update negative particles. The training similarly used a batch size of 128 samples of size 71×71. The missing positive phase boundary data for the latentBorders-method were outpainted by using block-Gibbs sampling for ten iterations starting from zero unit values within each iteration of the outer loop FPCD training algorithm. The learning rates for the different methods assumed a similar annealing scheme as in the previous subsection, but had different initial learning rates. Appropriate learning rates for the different methods were investigated experimentally, and here we report the results with the ones that obtained best results per approach. The NBoZ and the BoZ-methods used learning rates of 0.0005 for the weights, and 0.032 for the hidden unit biases. The LBo-method used 0.5 times the above learning rates. The SBi-method used learning rates of 0.0005 for all of the parameters.

C.1.2.3 Unconstrained Synthesis

Samples were collected by storing the conditional means of visible units given the hidden units of 128 independently and randomly initialized states after 100,000 iterations of Markov chain Monte Carlo for each analysis case. Figure C.8 shows example samples, and raw test data patches of similar size. The quality of the

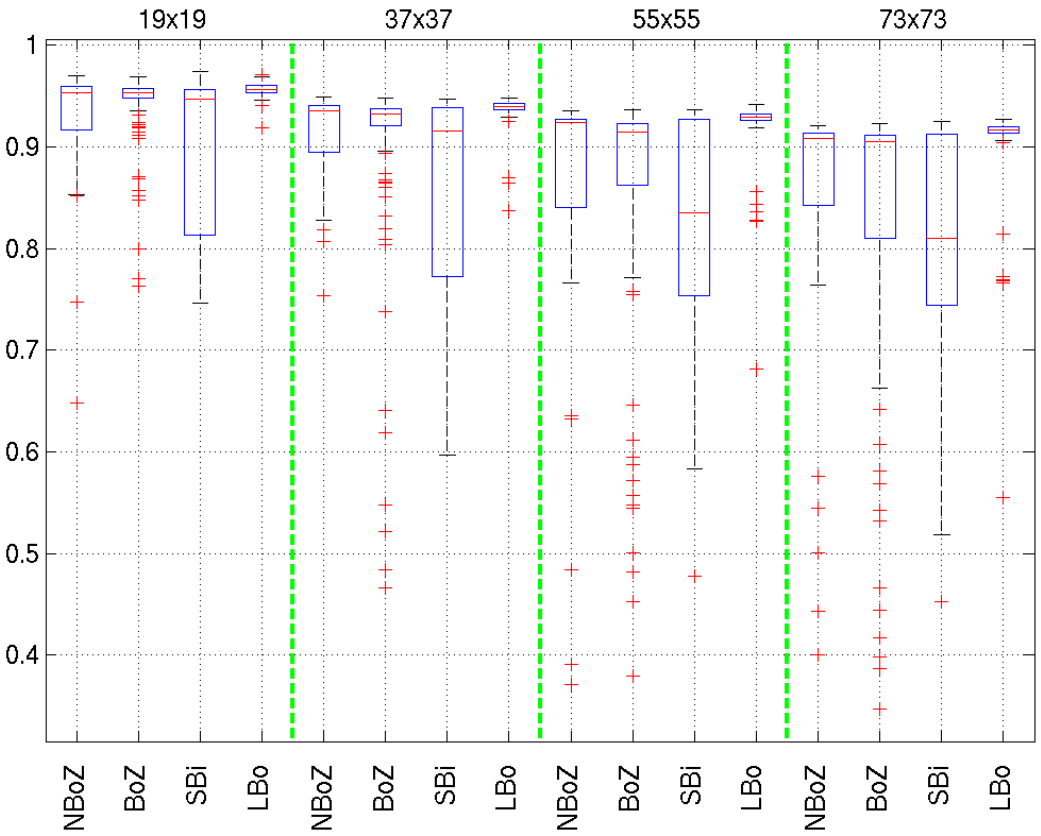


Figure C.6: D21 Brodatz texture synthesis quality assessment for the methods measured as texture similarity score between model samples and the testing half of the texture, using various window sizes in the score computation (horizontal blocks). Boxes indicate the upper and lower quartiles as well as the median (red bar) of the TSS distributions; whiskers show extent of the rest of the data; red crosses denote outliers.

samples were analyzed quantitatively by computing TSS-scores of the samples (with boundary data cropped out) to the test portions of the corresponding textures, at various matching window sizes. These results are summarized in Figures C.6 (D21), and C.7 (D77), and in Table C.3. From these summaries we can see that the latentBorders-method achieves the best results for both of the textures, and clearly so in the case of the D77 texture.

C.1.2.4 Constrained Synthesis

Inpainting performance was assessed on the 20 cases of previous subsection, each consisting of a 79×79 unit image, having 11-unit ground truth borders, and a 57×57 inpainting hole in the center. Each of the cases had a corresponding ground

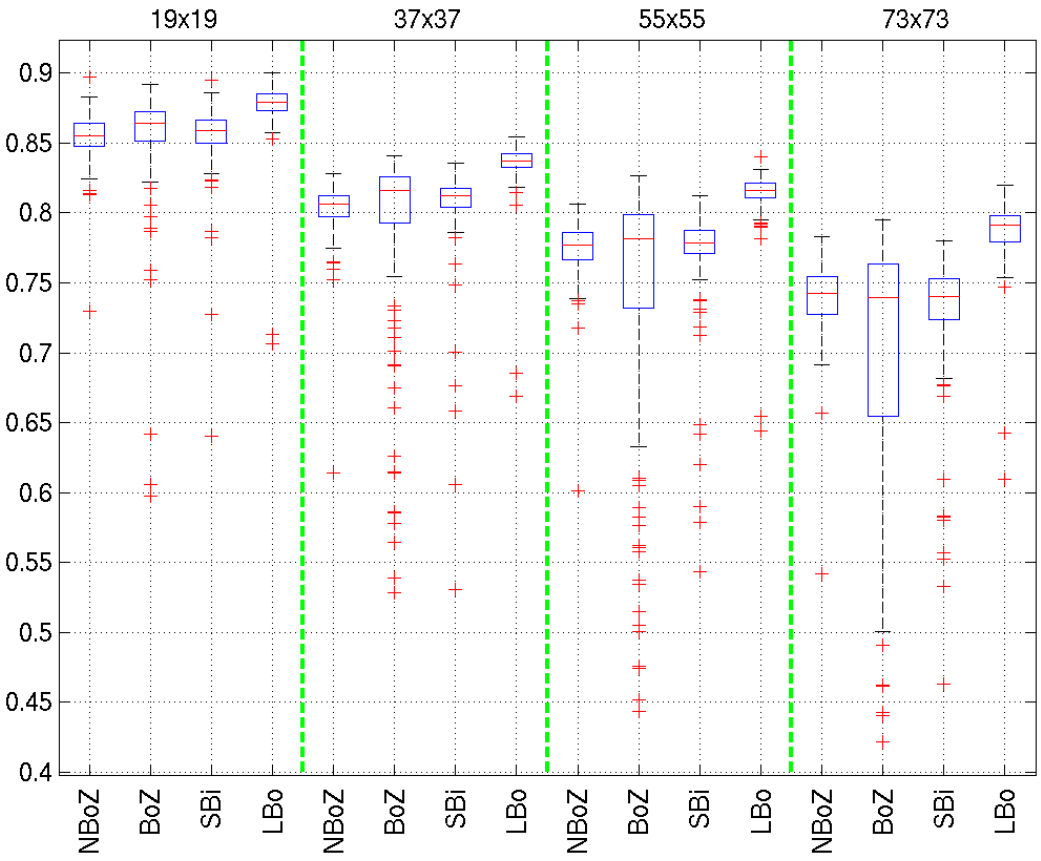


Figure C.7: D77 Brodatz texture synthesis quality assessment for the methods measured as texture similarity score between model samples and the testing half of the texture, using various window sizes in the score computation (horizontal blocks). Boxes indicate the upper and lower quartiles as well as the median (red bar) of the TSS distributions; whiskers show extent of the rest of the data; red crosses denote outliers.

Brodatz Texture D21 Synthesis (TSS):

	Boundary Handling Approach			
W.s.	negBordersZero	bordersZero	specialBiases	latentBorders
19×19	0.9328 ± 0.0460	0.9437 ± 0.0331	0.8931 ± 0.0735	0.9565 ± 0.0062
25×25	0.9272 ± 0.0361	0.9237 ± 0.0700	0.8802 ± 0.0782	0.9487 ± 0.0089
31×31	0.9117 ± 0.0684	0.9049 ± 0.0860	0.8669 ± 0.0820	0.9434 ± 0.0089
37×37	0.9135 ± 0.0420	0.9048 ± 0.0864	0.8557 ± 0.0880	0.9371 ± 0.0145
43×43	0.9002 ± 0.0642	0.8917 ± 0.0899	0.8520 ± 0.0841	0.9315 ± 0.0341
49×49	0.8912 ± 0.0721	0.8719 ± 0.1129	0.8437 ± 0.0951	0.9269 ± 0.0329
55×55	0.8820 ± 0.0916	0.8631 ± 0.1149	0.8366 ± 0.0943	0.9238 ± 0.0283
61×61	0.8801 ± 0.0824	0.8547 ± 0.1228	0.8310 ± 0.0949	0.9177 ± 0.0423
67×67	0.8721 ± 0.0933	0.8454 ± 0.1298	0.8251 ± 0.1008	0.9124 ± 0.0451
73×73	0.8673 ± 0.0895	0.8387 ± 0.1292	0.8213 ± 0.0976	0.9089 ± 0.0418

Brodatz Texture D77 Synthesis (TSS):

	Boundary Handling Approach			
W.s.	negBordersZero	bordersZero	specialBiases	latentBorders
19×19	0.8547 ± 0.0173	0.8532 ± 0.0439	0.8551 ± 0.0272	0.8770 ± 0.0227
25×25	0.8288 ± 0.0247	0.8227 ± 0.0536	0.8303 ± 0.0374	0.8601 ± 0.0216
31×31	0.8156 ± 0.0169	0.8123 ± 0.0518	0.8164 ± 0.0279	0.8453 ± 0.0171
37×37	0.8026 ± 0.0217	0.7905 ± 0.0675	0.8037 ± 0.0381	0.8347 ± 0.0216
43×43	0.7950 ± 0.0260	0.7751 ± 0.0747	0.7877 ± 0.0495	0.8271 ± 0.0219
49×49	0.7837 ± 0.0256	0.7597 ± 0.0845	0.7809 ± 0.0390	0.8187 ± 0.0223
55×55	0.7738 ± 0.0219	0.7434 ± 0.0919	0.7694 ± 0.0409	0.8123 ± 0.0227
61×61	0.7647 ± 0.0268	0.7305 ± 0.0893	0.7564 ± 0.0474	0.8043 ± 0.0232
67×67	0.7536 ± 0.0251	0.7179 ± 0.0923	0.7442 ± 0.0462	0.7956 ± 0.0229
73×73	0.7392 ± 0.0263	0.6990 ± 0.0941	0.7277 ± 0.0496	0.7865 ± 0.0242

Table C.3: Sample means and standard deviations of the texture synthesis TSS-scores for D21 (top) and D77 (bottom), under different methods (columns) and using different window sizes (W.s.) in the score computation (rows).

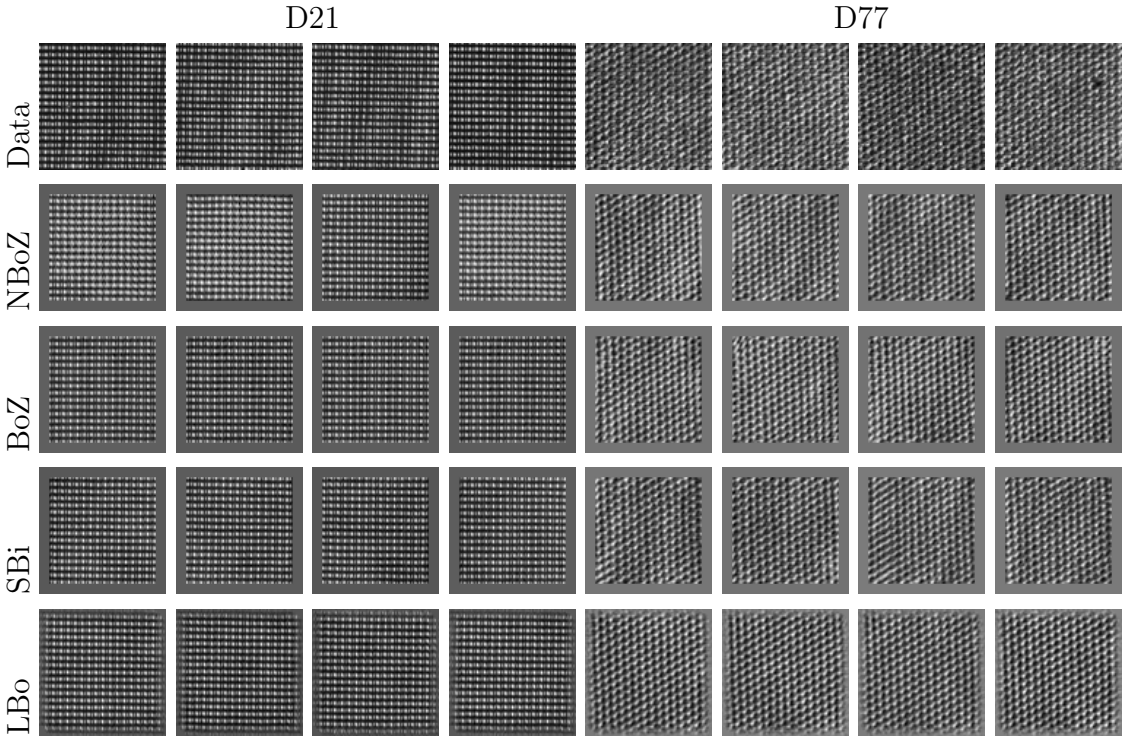


Figure C.8: Example data patches (top row), and method-specific model samples (other rows; see text for description), with each case scaled independently to cover the full intensity range. Borders are constrained to be zeros for all models, except for the latentBorder-method.

Brodatz Texture D21 Inpainting:

	Boundary Handling Approach			
Metric	negBordersZero	bordersZero	specialBiases	latentBorders
NCC	0.9154 ± 0.0118	0.9088 ± 0.0142	0.9162 ± 0.0115	0.9153 ± 0.0132
MSSIM	0.9081 ± 0.0154	0.8795 ± 0.0188	0.9097 ± 0.0148	0.9086 ± 0.0172
TSS	0.9246 ± 0.0055	0.9240 ± 0.0045	0.9253 ± 0.0047	0.9283 ± 0.0041

Brodatz Texture D77 Inpainting:

	Boundary Handling Approach			
Metric	negBordersZero	bordersZero	specialBiases	latentBorders
NCC	0.7644 ± 0.0198	0.7675 ± 0.0254	0.7294 ± 0.0276	0.7948 ± 0.0182
MSSIM	0.7603 ± 0.0185	0.7692 ± 0.0227	0.7337 ± 0.0276	0.7838 ± 0.0284
TSS	0.7886 ± 0.0099	0.7947 ± 0.0169	0.7609 ± 0.0180	0.8160 ± 0.0142

Table C.4: Sample means and standard deviations of the texture inpainting scores for D21 (top) and D77 (bottom), under different methods (columns) and performance metrics (rows).

truth texture patch within the test portion of the corresponding Brodatz-texture, and their positions were chosen randomly.

Inpainting was done for each case by applying block-Gibbs sampling with 5 chains started from the inpainting frames each associated with different random number generator seed for 9,999 iterations, sampling hidden units given visible units, and using the conditional means of visible units given the hidden units as the results. Figure C.10 shows example inpainting problems under both of the Brodatz textures, and results under each of the boundary handling methods.

To obtain quantitative results, normalized cross-correlation (NCC), and mean structural similarity index (MSSIM) were computed between inpainted region and the corresponding ground truth texture region, and texture similarity score between the inpainted region, and the test portion of the corresponding texture, for each sample. These results are summarized in Figure C.9, and in Table C.4. Similar to unconstrained synthesis, the latentBorders-method yields clearly best performance in the D77-texture case. In case of the D21-texture this approach performs similarly to the best of the rest.

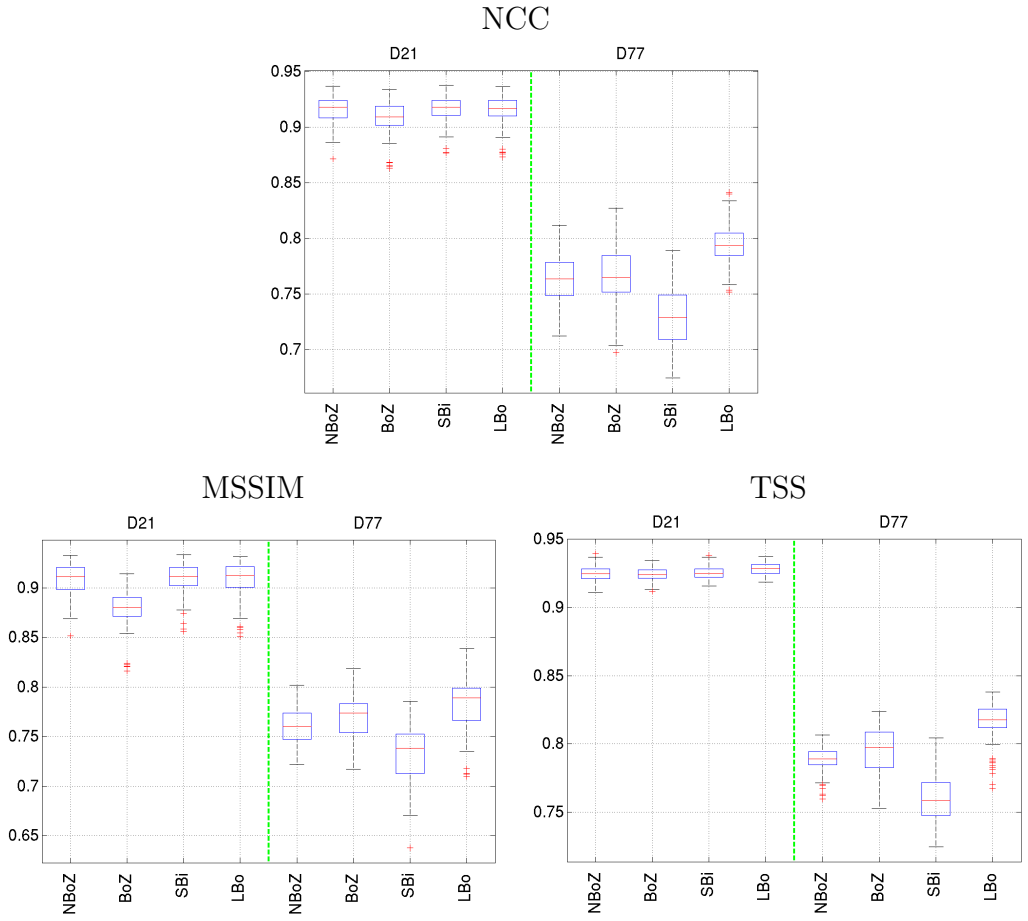


Figure C.9: Quality assessment for the boundary handling methods, based on NCC/MSSIM/TSS between inpainted area and corresponding Brodatz texture. Boxes indicate the upper and lower quartiles as well as the median (red bar) of the TSS/NCC distributions; whiskers show extent of the rest of the data; red crosses denote outliers.

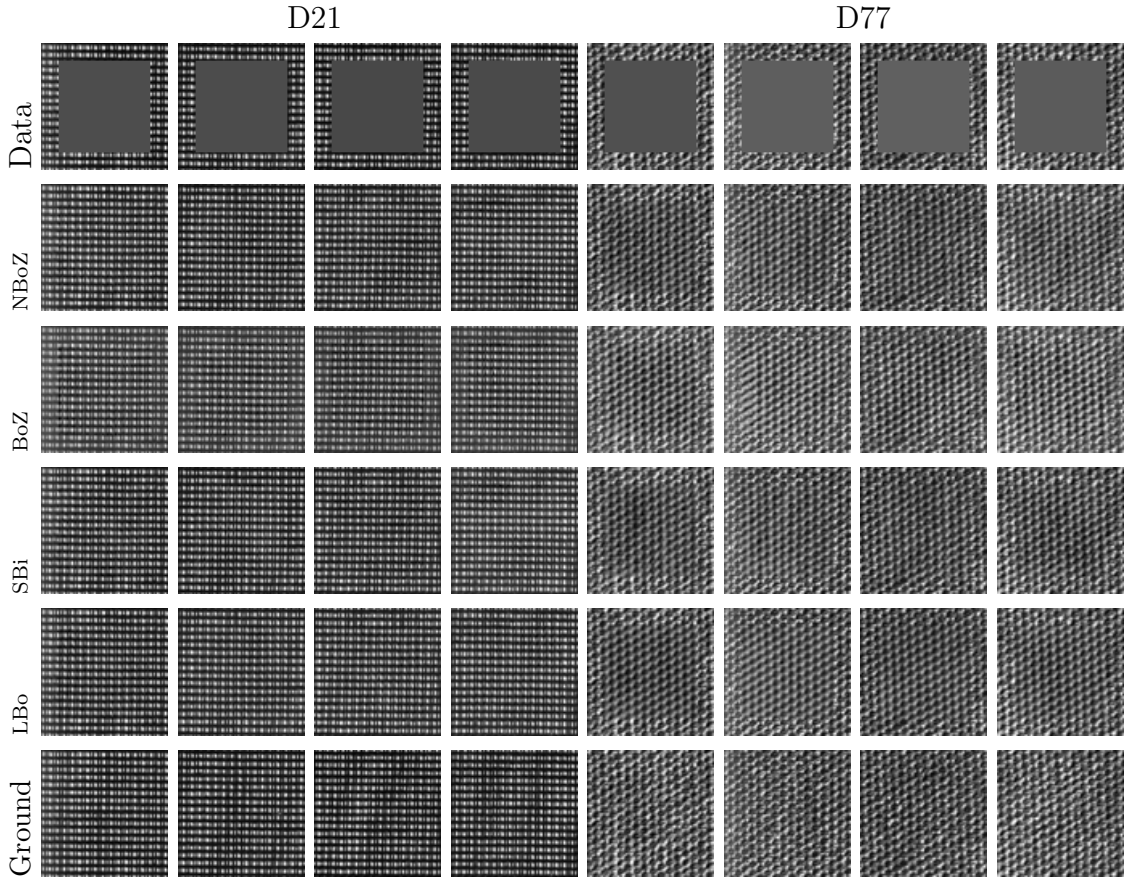


Figure C.10: Example inpainting frames (top row), ground truth (bottom row), and method-specific model samples (other rows; see text for description), with each case scaled independently to cover the full intensity range.

C.1.2.5 Discussion

Taking both the unconstrained and constrained texture synthesis quality into account, the latentBorders-method yields clearly the best results. Amongst the other approaches, the negBordersZero-method obtained overall best average performance, with the two other performing rather similarly on average across the textures, with mixed ranks for the two different textures.

The latentBorders-approach could be combined with a wrap-around, a commonly used method in the literature. While all of the units would be equally constrained in such approach, it is possible that the latent boundary would need to be much larger than considered here for obtaining effective performance, and even for very large such boundaries, the approach might be problematic for some textures. The problem is associated with obtaining a seamless texture pattern in an unfolded representation without distorting textons and their joint statistics.

C.1.3 Other considerations

In this this appendix and in Chapters 3 and 4 we have considered only single-channel input images (gray-scale texture images). It is straightforward to extend the methods to deal with multi-channel inputs, by using multi-channel weight kernels.

To add robustness to scale variations, and to reduce the amount of manual registration work, filter kernels at various sizes could be used to connect to visible units. Also in Chapter 4 tiled-convolutional feature sharing was used, whereas the methods in Chapter 3 considered non-tiled convolutional feature sharing. Currently there are no quantitative results suggesting the usefulness of one approach over the other in terms of generative performance. It could be the case that this would be dependent on the data to be modelled. A data-adaptive approach could be obtained by utilizing the transformation-equivariant modelling framework, as described in Chapter 6.

C.2 Partial derivatives in learning the texture models

Free-energies under the tiled-convolutional single-texture Tm, TPoT, and TmPoT models can be written as:

$$F_{\mathrm{Tm}}(\mathbf{v}) = \sum_i \frac{(v_i - a)^2}{2\sigma^2} - \sum_s \sum_t \sum_\ell \log\left\{1 + \exp\left\{b_\ell^s + \frac{1}{\sigma^2}\mathbf{M}_{\cdot\ell}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right\}\right\} \tag{C.1}$$

$$F_{\mathrm{TPoT}}(\mathbf{v}) = \sum_s \sum_t \sum_j \gamma_j^s \log\left\{1 + \frac{1}{2}\left[\mathbf{C}_{\cdot j}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right]^2\right\} \tag{C.2}$$

$$F_{\mathrm{TmPoT}}(\mathbf{v}) = F_{\mathrm{Tm}}(\mathbf{v}) + F_{\mathrm{TPoT}}(\mathbf{v}) \tag{C.3}$$

Let $F(\mathbf{v}) = F_{\mathrm{TmPoT}}(\mathbf{v})$, and $\sigma^2 = \exp\{z\}$. Partial derivatives of the free-energy with respect to the parameters can be written as follows:

$$\frac{\partial F(\mathbf{v})}{\partial a} = \sum_i \frac{(v_i - a)}{\sigma^2} \tag{C.4}$$

$$\frac{\partial F(\mathbf{v})}{\partial \mathbf{W}_{\cdot\ell}^s} = -\frac{1}{\sigma^2}\sum_t \frac{\exp\left\{b_\ell^s + \frac{1}{\sigma^2}\mathbf{M}_{\cdot\ell}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right\}}{1 + \exp\left\{b_\ell^s + \frac{1}{\sigma^2}\mathbf{M}_{\cdot\ell}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right\}}\mathbf{v}_{\mathcal{N}_{s,t}} \tag{C.5}$$

$$\frac{\partial F(\mathbf{v})}{\partial b_\ell^s} = -\sum_t \frac{\exp\left\{b_\ell^s + \frac{1}{\sigma^2}\mathbf{M}_{\cdot\ell}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right\}}{1 + \exp\left\{b_\ell^s + \frac{1}{\sigma^2}\mathbf{M}_{\cdot\ell}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right\}} \tag{C.6}$$

$$\frac{\partial F(\mathbf{v})}{\partial z} = -\frac{1}{\sigma^2}\left[\sum_i (v_i - a)^2 - \sum_s \sum_t \sum_\ell \frac{\exp\left\{b_\ell^s + \frac{1}{\sigma^2}\mathbf{M}_{\cdot\ell}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right\}}{1 + \exp\left\{b_\ell^s + \frac{1}{\sigma^2}\mathbf{M}_{\cdot\ell}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right\}}\mathbf{M}_{\cdot\ell}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right] \tag{C.7}$$

$$\frac{\partial F(\mathbf{v})}{\partial \mathbf{C}_{\cdot j}^s} = \sum_t \frac{\gamma_j^s \mathbf{C}_{\cdot j}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}}{1 + \frac{1}{2}\left[\mathbf{C}_{\cdot j}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right]^2} \tag{C.8}$$

$$\frac{\partial F(\mathbf{v})}{\partial \gamma_j^s} = \sum_t \log\left\{1 + \frac{1}{2}\left[\mathbf{C}_{\cdot j}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right]^2\right\} \tag{C.9}$$

Partial derivatives of the free-energies with respect to a visible unit v_i are the following:

$$\frac{\partial F_{\mathrm{Tm}}(\mathbf{v})}{\partial v_i} = \frac{v_i - a}{\sigma^2} - \frac{1}{\sigma^2}\sum_{s,t:v_i \in \mathbf{v}_{\mathcal{N}_{s,t}}} \sum_\ell \frac{\exp\left\{b_\ell^s + \frac{1}{\sigma^2}\mathbf{M}_{\cdot\ell}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right\}}{1 + \exp\left\{b_\ell^s + \frac{1}{\sigma^2}\mathbf{M}_{\cdot\ell}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right\}}\mathbf{M}_{t\to i,\ell}^s \tag{C.10}$$

$$\frac{\partial F_{\mathrm{TPoT}}(\mathbf{v})}{\partial v_i} = \sum_{s,t:v_i \in \mathbf{v}_{\mathcal{N}_{s,t}}} \sum_j \frac{\gamma_j^s \mathbf{C}_{\cdot j}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}} C_{t\to i,j}^s}{1 + \frac{1}{2}\left[\mathbf{C}_{\cdot j}^{s\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right]^2} \tag{C.11}$$

$$, \tag{C.12}$$

where $\mathbf{M}^{s}_{t\to i,\ell}$ and $\mathbf{C}^{s}_{t\to i,j}$ denote the connection weights in $\mathbf{M}^{s}_{\cdot\ell}$ and in $\mathbf{C}^{s}_{\cdot j}$ associated with position s,t to visible unit v_i, respectively. Finally, we have that

$$\frac{\partial F_{\mathrm{TmPoT}}(\mathbf{v})}{\partial v_i} = \frac{\partial F_{\mathrm{Tm}}(\mathbf{v})}{\partial v_i} + \frac{\partial F_{\mathrm{TPoT}}(\mathbf{v})}{\partial v_i}. \tag{C.13}$$

Appendix D

Modelling Natural Images and their Contours

D.1 Gradient-based Learning

D.1.1 Partial derivatives in learning an mcRBM

The free-energy under a tiled-convolutional mcRBM (TmcRBM) in terms of the covariance part is the following:

$$F^c(\mathbf{v}) = -\sum_{s=1}^{S}\sum_{t=1}^{T}\sum_{j=1}^{J}\log\left\{1+\exp\left\{d_j^s - \frac{1}{2}\sum_f \pi_{jf}^s \left[{\mathbf{K}_{\cdot f}^s}^\top \mathbf{A}\mathbf{v}_{\mathcal{N}_{s,t}}\right]^2\right\}\right\}, \quad \text{(D.1)}$$

where $\mathbf{v}_{\mathcal{N}_{s,t}}$ denotes visible units in the tile with index t under a shift index s. There are S sets of parameters each containing J covariance unit biases, F factor filters, and a pooling matrix π^s of size $J \times F$. These are shared at T different locations, and assuming square images with $T^{1/2} \times T^{1/2}$ tile grid size, the image lattice is of size $T^{1/2}D + 2(D/2-1) \times T^{1/2}D + 2(D/2-1)$, where D is the even vertical and horizontal dimension of a square filter under the model. Partial derivatives of the free-energy with respect to the model parameters, and data are

thus the following:

$$\frac{\partial F^c(\mathbf{v})}{\partial d_j^s} = -\sum_{t=1}^{T} \mathcal{S}(\mathrm{R}_t^{s,j}) \tag{D.2}$$

$$\frac{\partial F^c(\mathbf{v})}{\partial \mathbf{K}_{\cdot f}^s} = \sum_{t=1}^{T} \mathbf{A}\mathbf{v}_{\mathcal{N}_{s,t}} \sum_{j=1}^{J} \mathcal{S}(\mathrm{R}_t^{s,j}) \pi_{jf}^s \left[{\mathbf{K}_{\cdot f}^s}^\top \mathbf{A}\mathbf{v}_{\mathcal{N}_{s,t}} \right] \tag{D.3}$$

$$\frac{\partial F^c(\mathbf{v})}{\partial \pi_{jf}^s} = \frac{1}{2} \sum_{t=1}^{T} \mathcal{S}(\mathrm{R}_t^{s,j}) {\mathbf{K}_{\cdot f}^s}^\top \mathbf{A}\mathbf{v}_{\mathcal{N}_{s,t}} \tag{D.4}$$

$$\frac{\partial F^c(\mathbf{v})}{\partial v_i} = \mathbf{A} \sum_{s,t:v_i \in \mathbf{v}_{\mathcal{N}_{s,t}}} \sum_{j=1}^{J} \mathcal{S}(\mathrm{R}_t^{s,j}) \sum_{f=1}^{F} \pi_{jf}^s \left[{\mathbf{K}_{\cdot f}^s}^\top \mathbf{A}\mathbf{v}_{\mathcal{N}_{s,t}} \right] \mathbf{K}_{s,t\to f}^s, \tag{D.5}$$

where $\mathrm{R}_t^{s,j} = d_j^s - \frac{1}{2} \sum_f \pi_{jf}^s \left[{\mathbf{K}_{\cdot f}^s}^\top \mathbf{A}\mathbf{v}_{\mathcal{N}_{s,t}} \right]^2$, $\mathcal{S}(x) = 1/(1 + \exp\{-x\})$, the logistic function, and $\mathbf{K}_{t\to i,f}^s$ denotes the connection weight in $\mathbf{K}_{\cdot f}^s$ associated with position s, t to visible unit v_i.

D.1.2 Partial derivatives in learning a logistic regression network for contour prediction

The training of the network was based on a stochastic gradient ascent algorithm, optimizing a function

$$\mathcal{C} = \mathcal{L} - \lambda R, \tag{D.6}$$

where $\mathcal{L}$ denotes the log-likelihood of the training data, where

$$\mathcal{L} = \sum_{n,i} \left[y_i^{(n)} \log u_i^{(n)} + (1 - y_i^{(n)}) \log\{1 - u_i^{(n)}\} \right], \tag{D.7}$$

where $y_i^{(n)}$ denotes the state of the contour unit with index i in training image with index n, and $u_i^{(n)}$ denotes the prediction with the method for that unit, according to equation (5.4). The other term is a regularization term $R = \frac{1}{2}\|\theta\|_2^2$, where λ is a positive regularization constant, and θ denotes the model parameters.

Let us first consider the partial derivative of $\mathcal{L}$ with respect to the parameters θ:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{n,i} \left[\frac{y_i^{(n)}}{u_i^{(n)}} \frac{\partial u_i^{(n)}}{\partial \theta} - \frac{(1 - y_i^{(n)})}{1 - u_i^{(n)}} \frac{\partial u_i^{(n)}}{\partial \theta} \right] \tag{D.8}$$

$$= \sum_{n,i} \left[\frac{y_i^{(n)} - u_i^{(n)}}{u_i^{(n)} \left(1 - u_i^{(n)}\right)} \frac{\partial u_i^{(n)}}{\partial \theta} \right] \tag{D.9}$$

$$\tag{D.10}$$

Let $\mathcal{S}(x) = 1/(1+\exp\{-x\})$. We can then write that $u_i^{(n)} = \mathcal{S}\left(g + \sum_j W_{ji} f_j^{(n)}\right) = \mathcal{S}(A_{i,n})$. As $\frac{\partial u_i^{(n)}}{\partial \theta} = \frac{\partial \mathcal{S}(A_{i,n})}{\partial \theta} = \mathcal{S}(A_{i,n})\left(1 - \mathcal{S}(A_{i,n})\right)\frac{\partial A_{i,n}}{\partial \theta} = u_i^{(n)}\left(1 - u_i^{(n)}\right)\frac{\partial A_{i,n}}{\partial \theta}$, we find by substitution and simplification that

$$\frac{\partial \mathcal{L}}{\partial \theta} = \left[\sum_{n,i}\left(y_i^{(n)} - u_i^{(n)}\right)\frac{\partial A_{i,n}}{\partial \theta}\right], \tag{D.11}$$

where $A_{i,n} = g + \sum_j W_{ji} f_j^{(n)}$. As $\frac{\partial R}{\partial \theta_k} = \theta_k$, we find that

$$\frac{\partial \mathcal{C}}{\partial W_{ji}} = \left[\sum_{n}\left(y_i^{(n)} - u_i^{(n)}\right) f_j^{(n)}\right] - \lambda W_{ji} \tag{D.12}$$

$$\frac{\partial \mathcal{C}}{\partial g} = \left[\sum_{n,i}\left(y_i^{(n)} - u_i^{(n)}\right)\right] - \lambda g. \tag{D.13}$$

$$\tag{D.14}$$

D.1.2.1 Back-propagating errors for adjusting network parameters

Let ϕ denote the parameters of a TmcRBM. The partial derivatives of the log-likelihood w.r.t. the parameters

$$\frac{\partial \mathcal{L}}{\partial \phi} = \left[\sum_{n,i}\left(y_i^{(n)} - u_i^{(n)}\right)\frac{\partial A_{i,n}}{\partial \phi}\right] \tag{D.15}$$

$$= \sum_{n,i}\left(y_i^{(n)} - u_i^{(n)}\right)\sum_j W_{ji}\frac{\partial f_j^{(n)}}{\partial \phi}. \tag{D.16}$$

As the hidden unit activations $f_j^{(n)}$ are defined using sigmoidal functions, we can write $f_j^{(n)} = \mathcal{S}(B_j^{(n)})$, and thus

$$\frac{\partial \mathcal{L}}{\partial \phi} = \sum_{n,i}\left(y_i^{(n)} - u_i^{(n)}\right)\sum_j W_{ji} f_j^{(n)}\left(1 - f_j^{(n)}\right)\frac{\partial B_j^{(n)}}{\partial \phi}. \tag{D.17}$$

For deeper networks the computation recurses in a similar fashion, proceeding from the last layer to the first layer, applying the chain-rule of differentiation for partial derivatives for parameters earlier in the networks. Note that a lot of computation is typically shared, and it would be naive to compute each partial derivative independently of other ones. The thesis uses computation sharing as in the back-propagation algorithm/formalization of the partial derivative computations [Rumelhart et al., 1986], with computations done in a layer-sweeping manner.

D.2 Metrics in model learning

In the main text, the models were learned by optimizing (ignoring the regularization terms) the discriminative log-likelihood of the training data, in which it is assumed that each point in the contour unit lattice is distributed according to a Bernoulli distribution, with success probability given according to the prediction (probability of contours) with the approach (i.e. $p(y_i = z_i|u_i) = u_i^{z_i}(1-u_i)^{1-z_i}$). Alternatively it can be viewed as optimizing the cross-entropy between the distribution of contour state and the probability of contour according to the prediction model:

$$H(p_i.q_i) = -\sum_s p_i(s) \log q_i(s), \tag{D.18}$$

where $p_i = [1 - y_i \; y_i]$, and $q_i = [1 - u_i \; u_i]$, where y_i denotes the label of contour unit with index i, and u_i is given by equation (5.4).

For any given image there are annotations by several humans, with binary decisions for the existence of contour under any image position. As Hou et al. [2013] discuss, a large amount of the positions marked containing a contour are from only one of the annotators. To avoid issues this causes under the benchmark metric, the paper proposes to consider the detection of strong contour positions (where there is a consensus among all of the annotators) as opposed to any contours.

In the following we are considering learning averages of the annotations per position. This could be related to the probability that humans consider the position to be a contour-position. In comparison to the strong-contour detection, this approach does not throw away annotation data as much. In both approaches, imperfections are expected due to alignment issues.

The benchmark metric ignores spatial dependencies, as each pixel is evaluated independent of the other ones[1], which could be more problematic than noisy annotation data. There are fidelity metrics for image data which take spatial coherence into account. One such is the structural similarity index [Wang et al., 2004] (see also Appendix A.5), which is widely used for example in the image restoration community.

Building on these observations and ideas, we consider in the following the

[1]The metric declares a predicted edge correct if there exists an edge within any of the annotations at or around the prediction location. The metric does not consider prediction sites jointly, and so spatial coherence/continuation within the prediction images, the annotations, and their differences are not assessed.

task of predicting the probability for active contour annotation, and evaluate it according to the mean structural similarity index (MSSIM). We first assessed the performance for models optimized using the binary training data (optimizing a function consisting of the the discriminative log-likelihood and regularization terms). We then trained the two-stream model based on the probability images, optimizing a function consisting of regularization terms and cross-entropy as above but setting $p_i = [1 - \mu_i \ \mu_i]$, where μ_i is the average of the A label annotations for position i i.e. $\mu_i = \frac{1}{A}\sum_a y_i^a$. Note that if in training each batch contains all of the available annotations for each image data, the approaches are related by scalings (but using the average makes the training faster).

Tables D.1 and D.2 summarize the results for the BSDS500 test set under the shallow and deeper models, respectively. Figure D.1 visualizes the MSSIM scores for the 200 test images, sorted according to the score, for shallow (top) and deeper models (bottom), and shows performance summaries in the form of average MSSIM. We observe that ordering of the approaches remains effectively the same in comparison to the results using the benchmark metrics, except for the two-stream model with 200 versus 300 training images, where the former performs better than the latter, as opposed to the other way around using the benchmark metrics. The models trained with probabilities are best also according to the MSSIM-based metrics, being there markedly best. The latter two properties can be also seen in Figure D.2, which compares the prediction results with for two-stream model, trained with either method, using either the benchmark P-R curves (top), or the MSSIM-score based curves (bottom). To summarize, we have proposed a novel benchmarking task of contour probability prediction and an evaluation method for assessing the performance. The performance evaluation uses the structural similarity index as the quality metric. We have considered the approach under the BSDS500 test data, setting the ground-truth contour probability images as the averages of the image-specific binary annotation results, evaluating with all of the models in the main chapter.

We found that the model performance orderings under the large set of models using the MSSIM approach were similar to those when using the BSDS benchmark evaluation methods. Although the evaluation metrics are expected to be in general complementary, there are some other expected advantages of the MSSIM approach over the benchmark evaluation. For example, the SSIM-metric takes spatial coherence into account, in contrast to the benchmark metric, and the

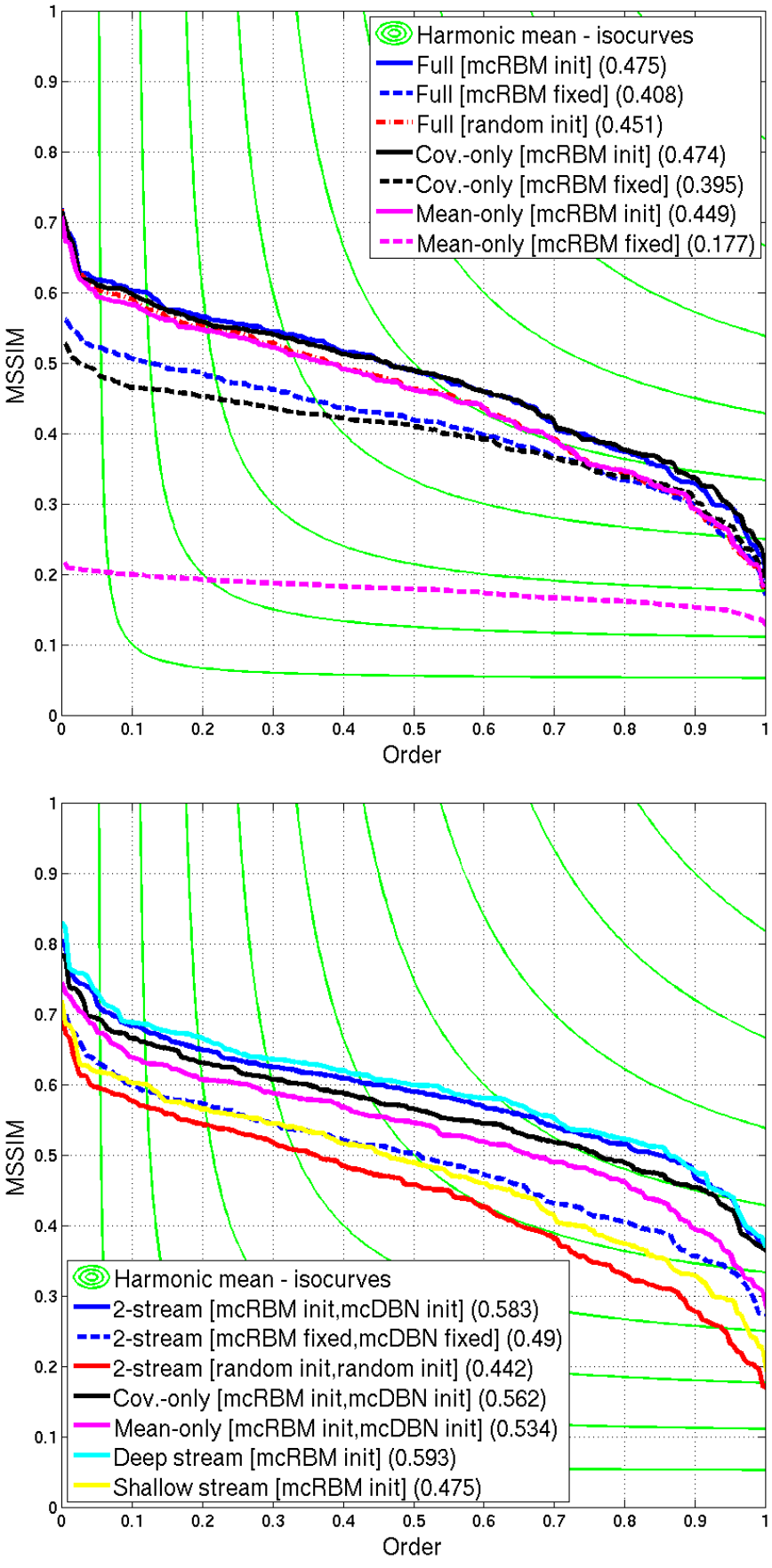


Figure D.1: Contour *probability* prediction results on the BSDS500 as measured by curves based on sorted MSSIM-scores, with shallow models (top), and with deeper models (bottom). The numbers in the legends denote the average MSSIMs.

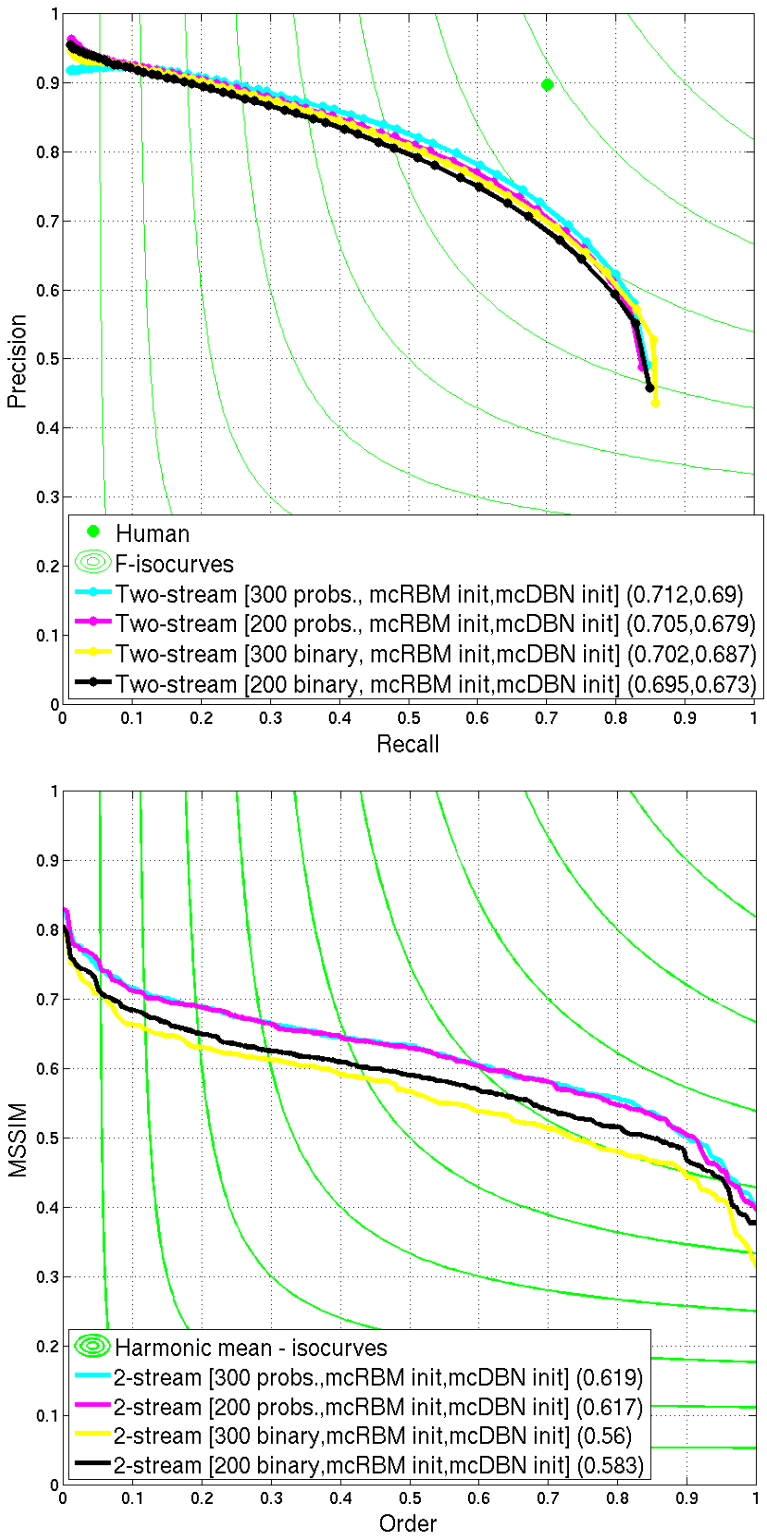


Figure D.2: Contour *existence (top) and probability (bottom)* prediction results on the BSDS500 as measured by P-R curves (top) and curves based on sorted MSSIM-scores (bottom), with the two-stream model using different training sets. Top: The numbers in the legends denote the maximums of the curves w.r.t. the harmonic mean (left), and the average precision (right). Bottom: The numbers in the legends denote the average MSSIMs.

	MSSIM-based sample statistics			
Model	Min.	Mean ± St. Dev.	Median	Max.
Full [mcRBM fixed]	0.171	0.408 ± 0.084	0.419	0.565
Full [random init]	0.181	0.451 ± 0.114	0.463	0.719
Full [mcRBM init]	0.193	0.475 ± 0.107	0.490	0.720
Cov.-only [mcRBM fixed]	0.180	0.395 ± 0.068	0.411	0.532
Cov.-only [mcRBM init]	0.202	0.474 ± 0.102	0.490	0.715
Mean-only [mcRBM fixed]	0.128	0.177 ± 0.018	0.179	0.221
Mean-only [mcRBM init]	0.184	0.449 ± 0.111	0.462	0.709

Table D.1: Mean structural similarity index-based statistics on boundary *probability* prediction under the **BSDS500** test set by the **shallow** prediction model by using different feature sets. The feature extraction settings for the models are shown within the parentheses.

evaluation is expected to be computationally much less demanding .

It is noted that is is possible to tune the model parameters by optimizing the MSSIM-metric directly, as it is exactly differentiable, as opposed to the F-score (although approximations exist, and which have been used in this context for learning models, as in Kokkinos [2010]), but we did not consider such. One of the reasons for it is the following: although the metric has useful benefits as previously mentioned, it is still a proxy with also disadvantages, and expected not to be universally better than the other metrics, but to complement them.

D.3 Colour-Domain Model Results

Table D.3 compares the prediction results of our approach to the competing methods on the BSDS500 (colour) data set. As can be seen from the table, our approach yields state-of-the-art performance. The networks here had twice the number of hidden units compared to the networks for grey-scale data. They were trained on RGB-domain image data using a similar setup as for the gray-scale domain model but with two exceptions: (i) the target activation level for the mean hidden units in the colour-domain was 0.025 as opposed to 0.1 in the grey-scale domain, and (ii) learning rates were scaled to account for more input channels and also for more hidden units: the diagonally-tiled-convolutional mcRBM in the

Model	MSSIM-based sample statistics			
	Min.	Mean ± St. Dev.	Median	Max.
(Training on binary labels:)				
Two-stream [mcRBM fixed,mcDBN fixed]	0.274	0.490 ± 0.093	0.502	0.702
Two-stream [random init,random init]	0.170	0.442 ± 0.115	0.458	0.716
Two-stream [mcRBM init,random init]	0.294	0.508 ± 0.086	0.518	0.705
Two-stream [mcRBM init,mcDBN init]	0.377	0.583 ± 0.085	0.590	0.806
The above with 300 training images	0.315	0.560 ± 0.092	0.567	0.792
Deep stream [mcRBM fixed,mcDBN fixed]	0.244	0.477 ± 0.101	0.488	0.741
Deep stream [mcRBM init,mcDBN init]	0.370	0.593 ± 0.087	0.600	0.831
Cov.-only two-stream [mcRBM fixed,mcDBN init]	0.321	0.528 ± 0.084	0.537	0.732
Cov.-only two-stream [mcRBM init,mcDBN init]	0.365	0.562 ± 0.085	0.566	0.785
Cov.-only deep stream [mcRBM init,mcDBN init]	0.341	0.570 ± 0.090	0.579	0.825
Mean-only two-stream [mcRBM fixed,mcDBN init]	0.195	0.452 ± 0.106	0.456	0.713
Mean-only two-stream [mcRBM init,mcDBN init]	0.283	0.534 ± 0.095	0.546	0.745
Mean-only deep stream [mcRBM init,mcDBN init]	0.269	0.526 ± 0.095	0.536	0.734
Training on contour-strength labels:				
Two-stream [mcRBM init,mcDBN init]	0.3965	0.6172 ± 0.0863	0.6298	0.8292
The above with 300 training images	**0.4045**	$\mathbf{0.6191} \pm 0.0845$	**0.6334**	**0.8322**

Table D.2: Mean structural similarity index-based statistics on boundary *probability* prediction for the **BSDS500** (grey-scale) test set by the **deeper** prediction models by using different feature sets and architectures.

colour-domain had 1/6 the learning rates of the one in the grey-scale domain and the TmcDBN models were trained with a shared learning rate of 0.0001 as opposed to the 0.00025 in the grey-scale domain.

D.4 Boundary Prediction Result Comparison Examples

Figures D.3, D.4, D.5, D.6, D.7, and D.8 show BSDS500 test image data contour prediction examples with the Canny-method, the shallow-stream model, the deep-stream model, and the two-stream model.

D.5 Two-Stream Boundary Prediction Result Dissections

Figures D.9, D.10, D.11, D.12, D.13, D.14, D.15, D.16, D.17, and D.18 visualize BSDS500 test image data contour prediction examples with the two-stream model, and contributions of different components under the model to the predictions.

D.6 Generative models for joint representations of images and their contours

Here we develop generative models based on Boltzmann machines for joint generative models of natural image data, and their segment boundaries.

D.6.1 The Image and Contour Boltzmann Machine (icRBM)

We extend the (T)mcRBM by connecting the contour units to the hidden units for simplicity in such way that they enter each hidden unit specific term additively, and so the energy function (as a function of the hidden units $\mathbf{h}$, visible image

Model	F-score		AP
	ODS (P,R)	OIS (P,R)	
Ours:			
Two-stream (200)	0.729 (0.744,0.715)	0.748 (0.770,0.727)	0.698
Two-stream (300)	0.736 (0.735,0.737)	0.751 (0.757,0.745)	0.695
Two-stream (200, shift-averaging)	0.737 (0.756,0.719)	0.756 (0.744,0.756)	0.694
Two-stream (300, shift-averaging)	**0.744** (0.733,0.755)	**0.760** (0.755,0.765)	**0.692**
Others:			
SCG (global) [Ren and Bo, 2012]	**0.739** (0.754,0.725)	**0.758** (0.780,0.737)	**0.773**
gPb-owt-ucm [Arbelaez et al., 2011]	0.727 (0.726,0.726)	0.760 (0.768,0.751)	0.727
(Literature results where rounding had been done by 2 decimal places):			
Sketch-Tokens [Lim et al., 2013]	0.73	0.75	0.78
gPb (global) [Arbelaez et al., 2011] (from Ren and Bo [2012])	0.71	0.74	0.72

Table D.3: Contour prediction on the *colour-domain* BSDS500 dataset with various methods. Scores of our method and the highest ranking competitor are emphasized.

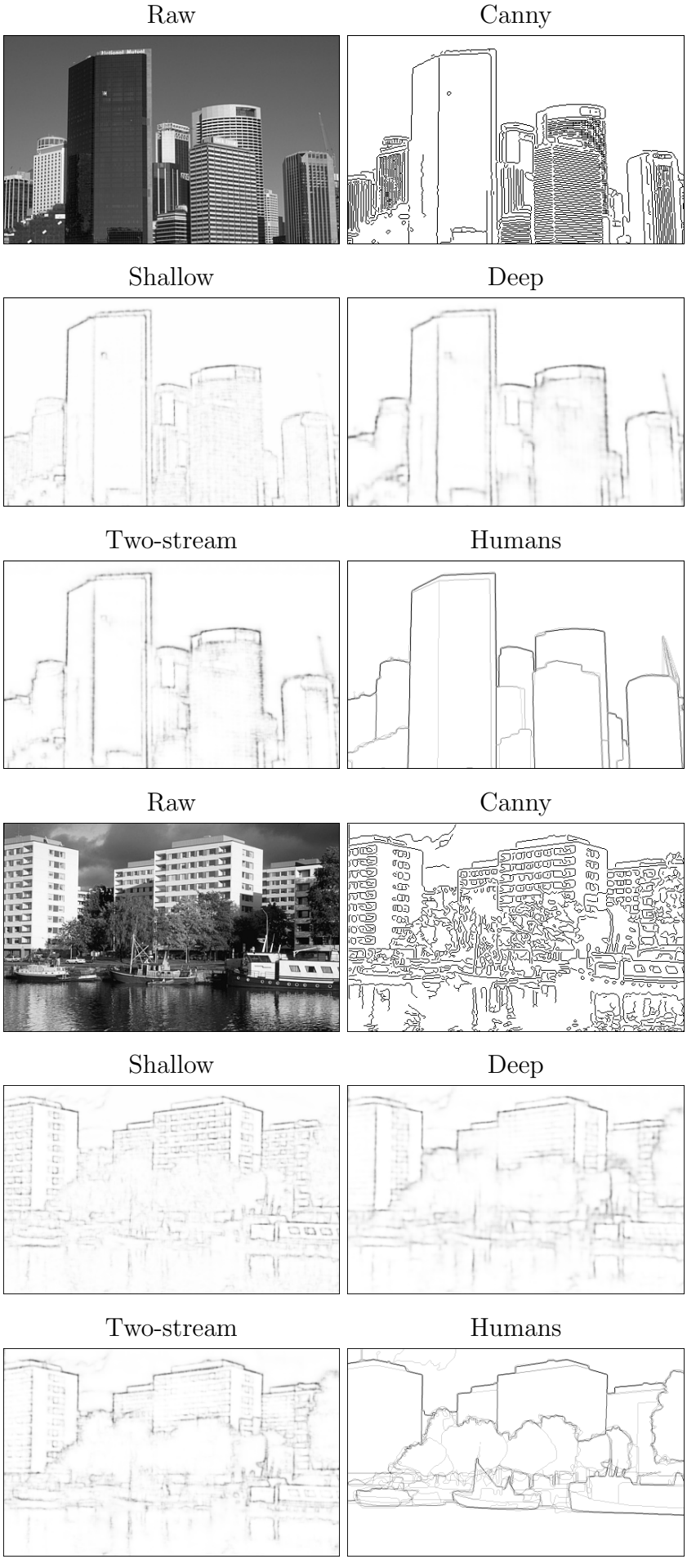


Figure D.3: Contour prediction result example on the grey-scale BSDS500. Best viewed on screen.

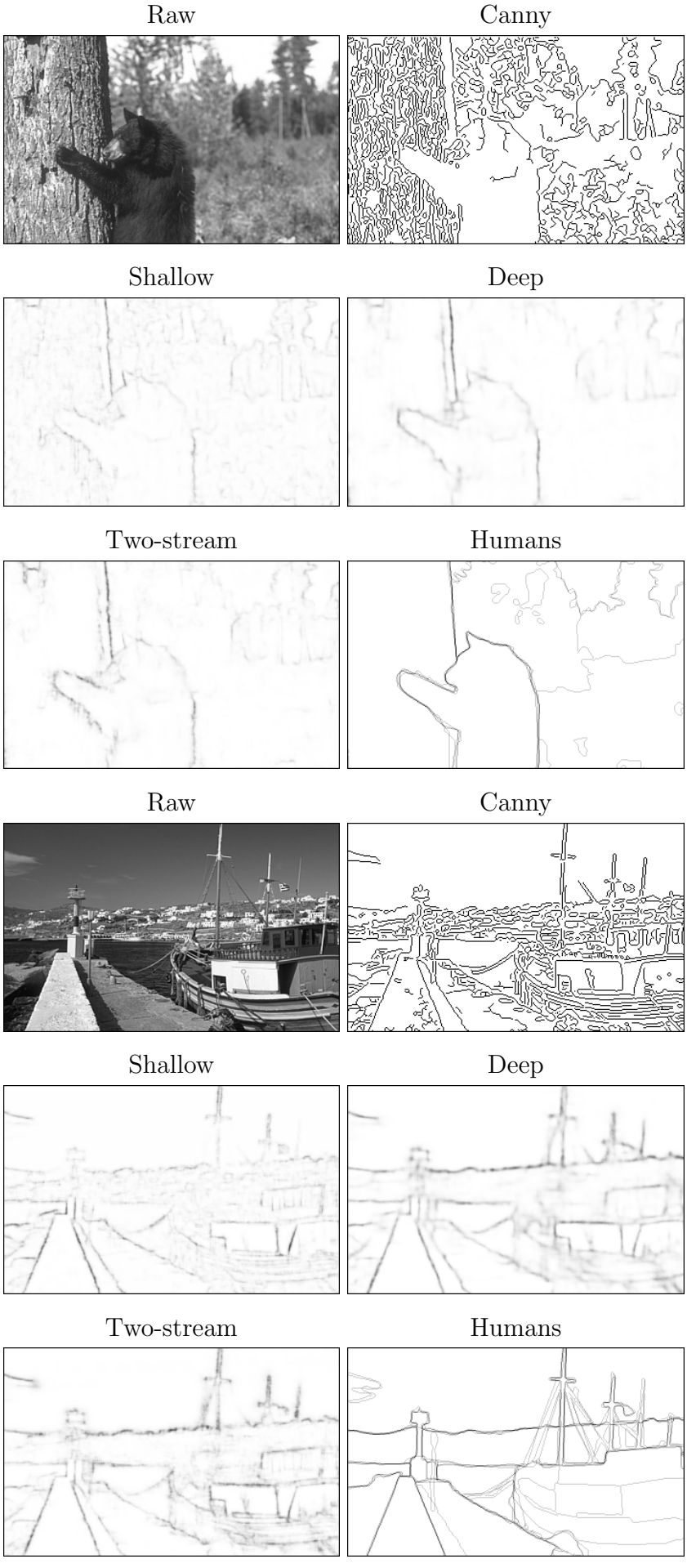


Figure D.4: Contour prediction result example on the grey-scale BSDS500. Best viewed on screen.

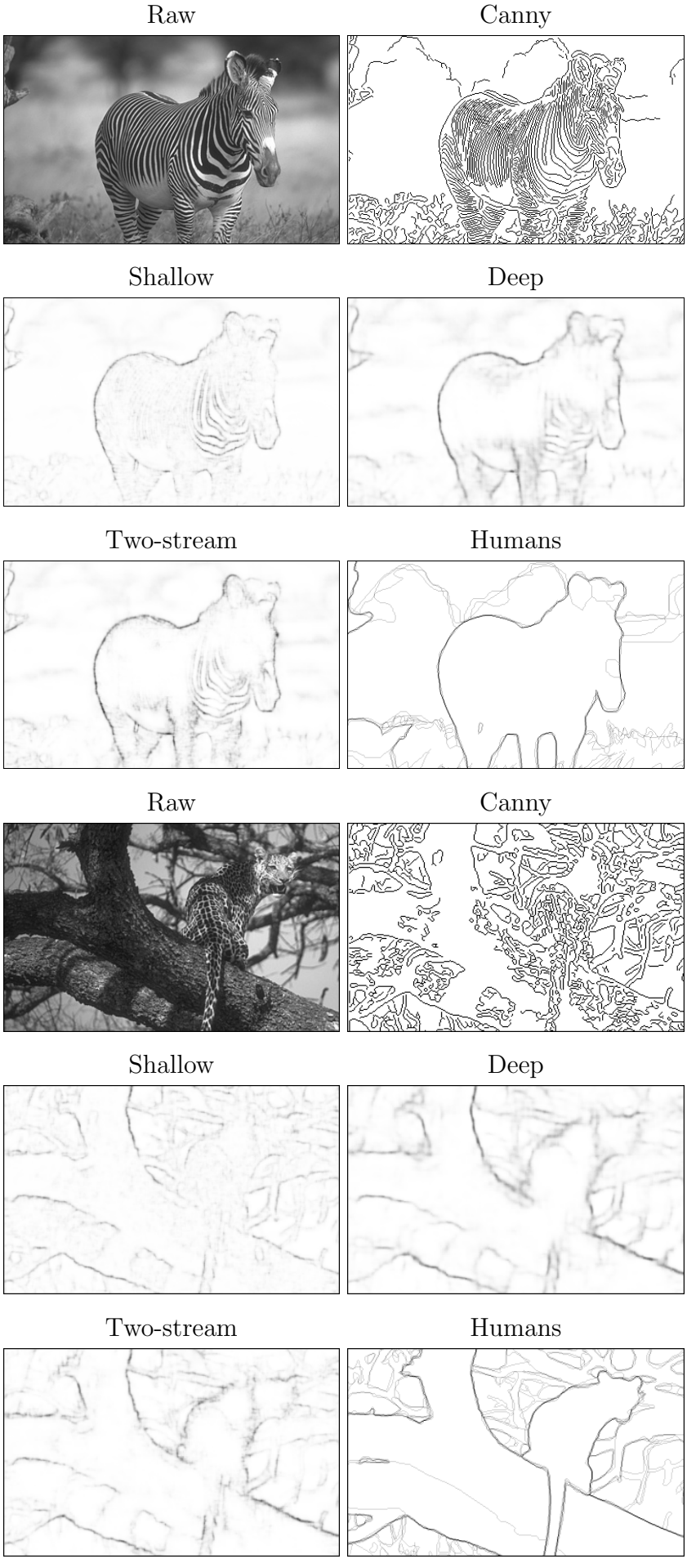


Figure D.5: Contour prediction result example on the grey-scale BSDS500. Best viewed on screen.

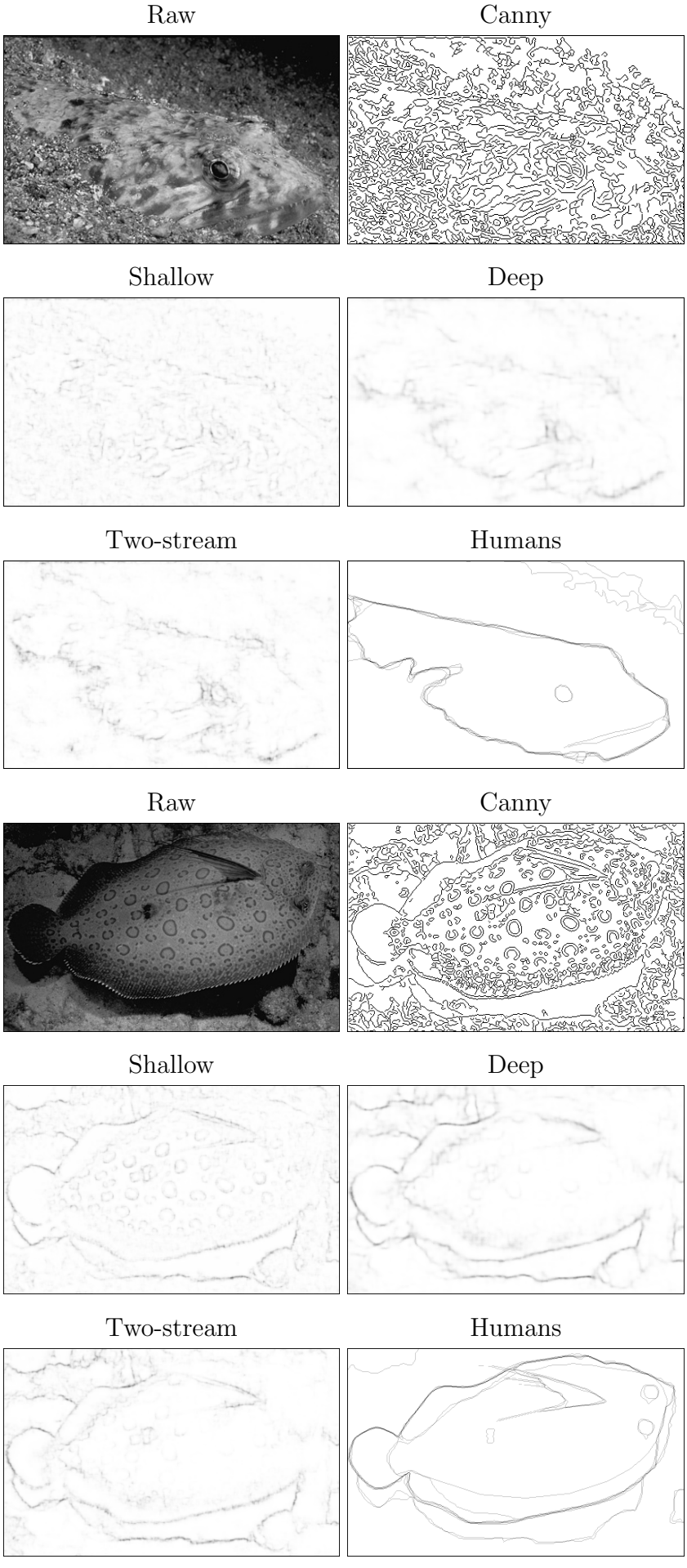


Figure D.6: Contour prediction result example on the grey-scale BSDS500. Best viewed on screen.

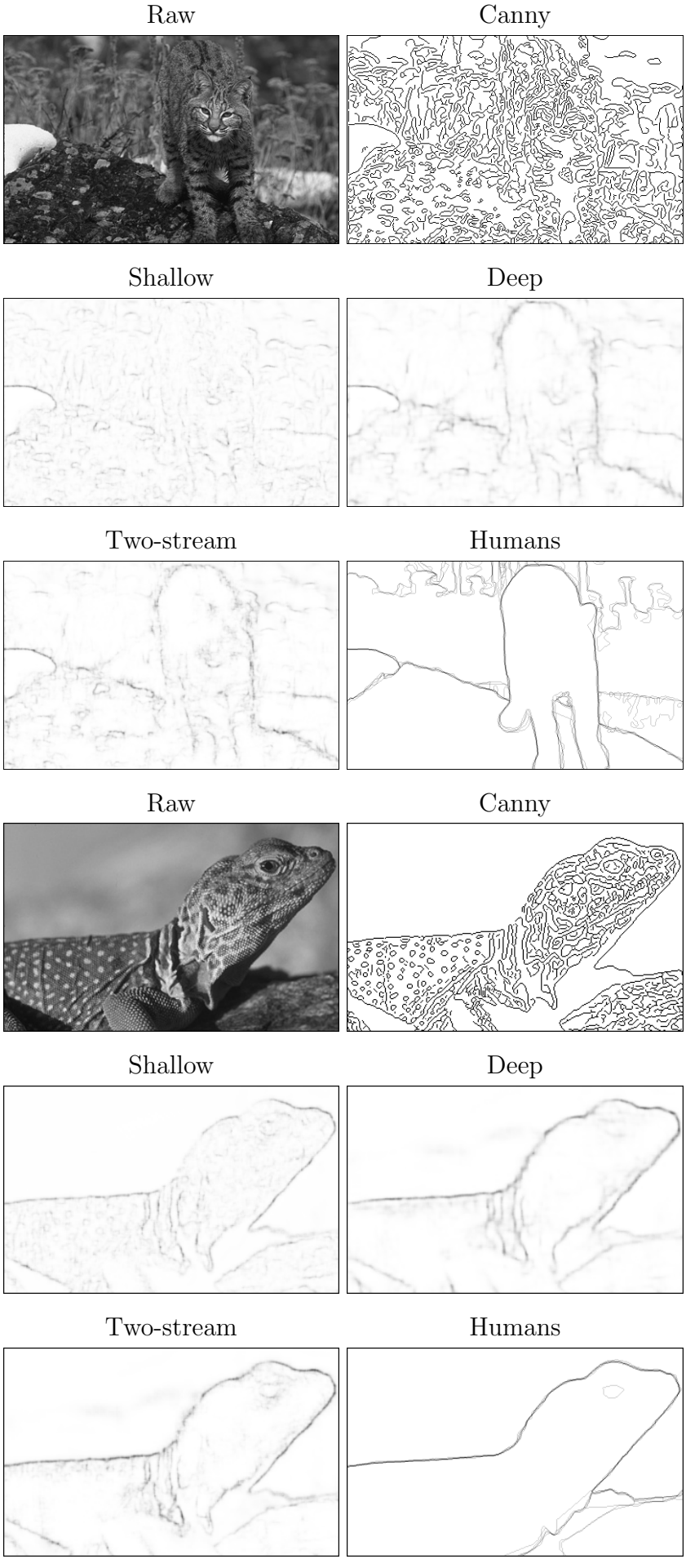


Figure D.7: Contour prediction result example on the grey-scale BSDS500. Best viewed on screen.

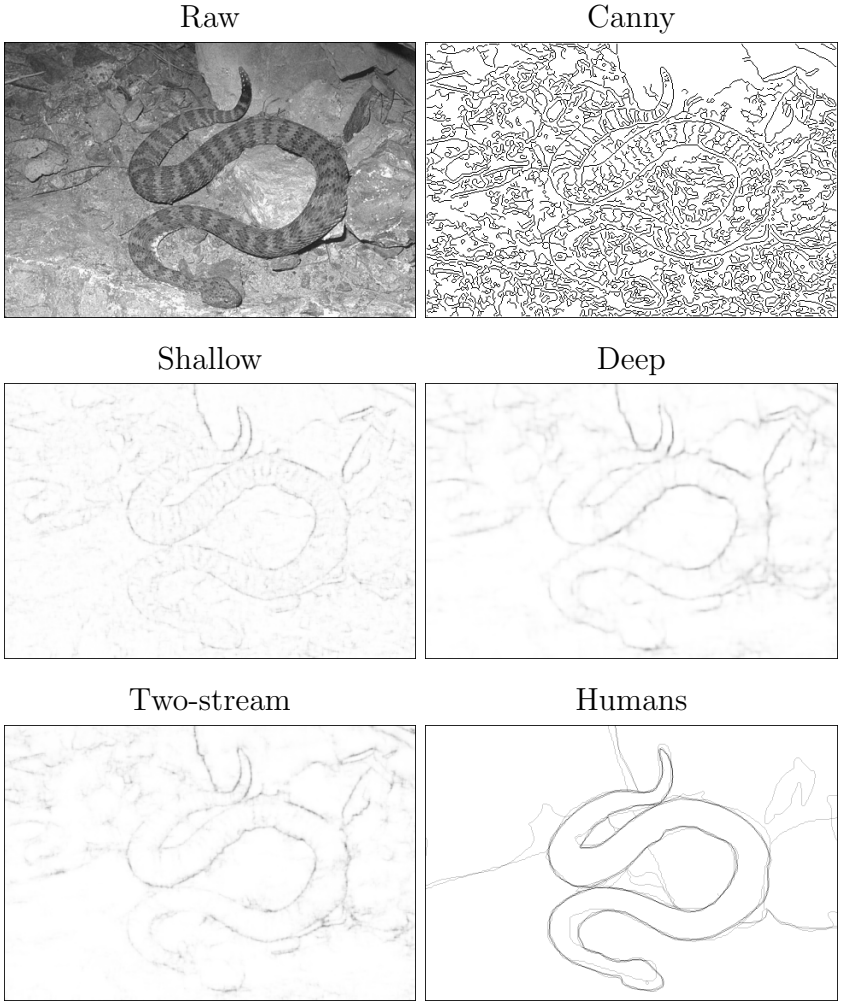


Figure D.8: Contour prediction result example on the grey-scale BSDS500. Best viewed on screen.

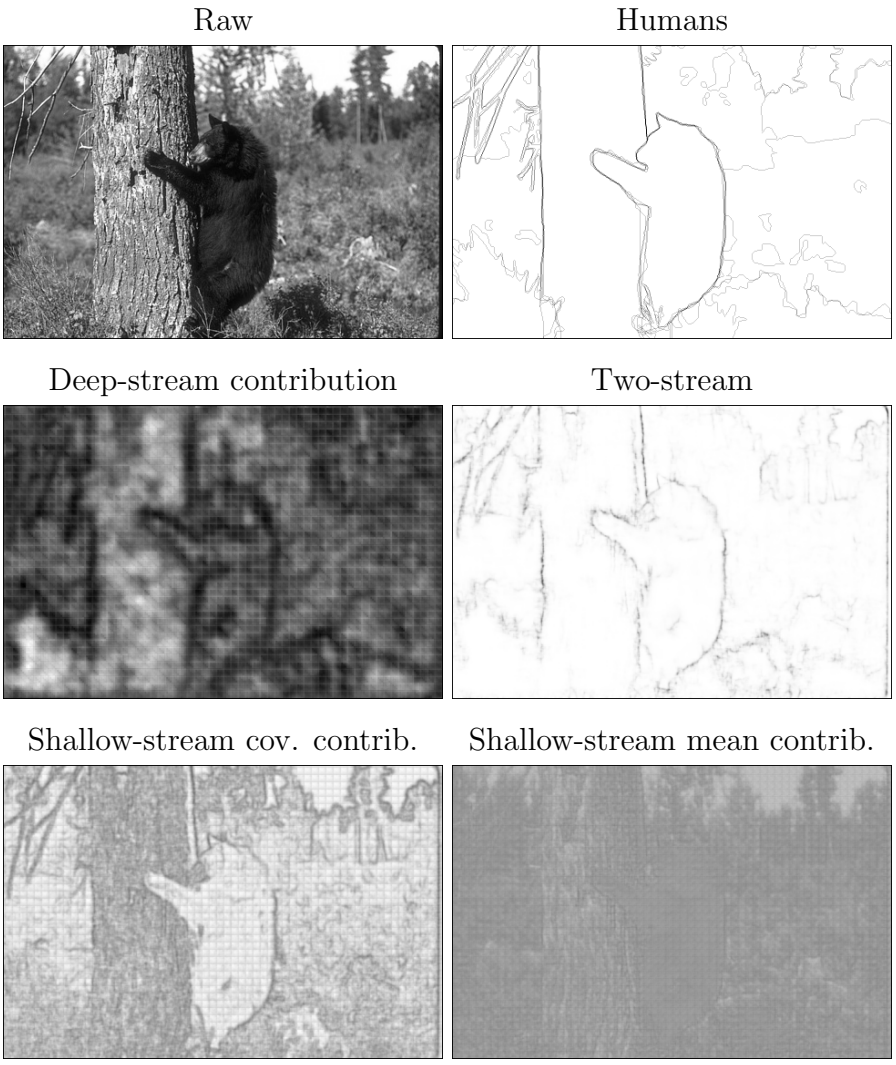


Figure D.9: Dissecting a contour prediction result on the grey-scale BSDS500. The stream contributions are normalized to be on the same scale, which also fills the full intensity range. Best viewed on screen.

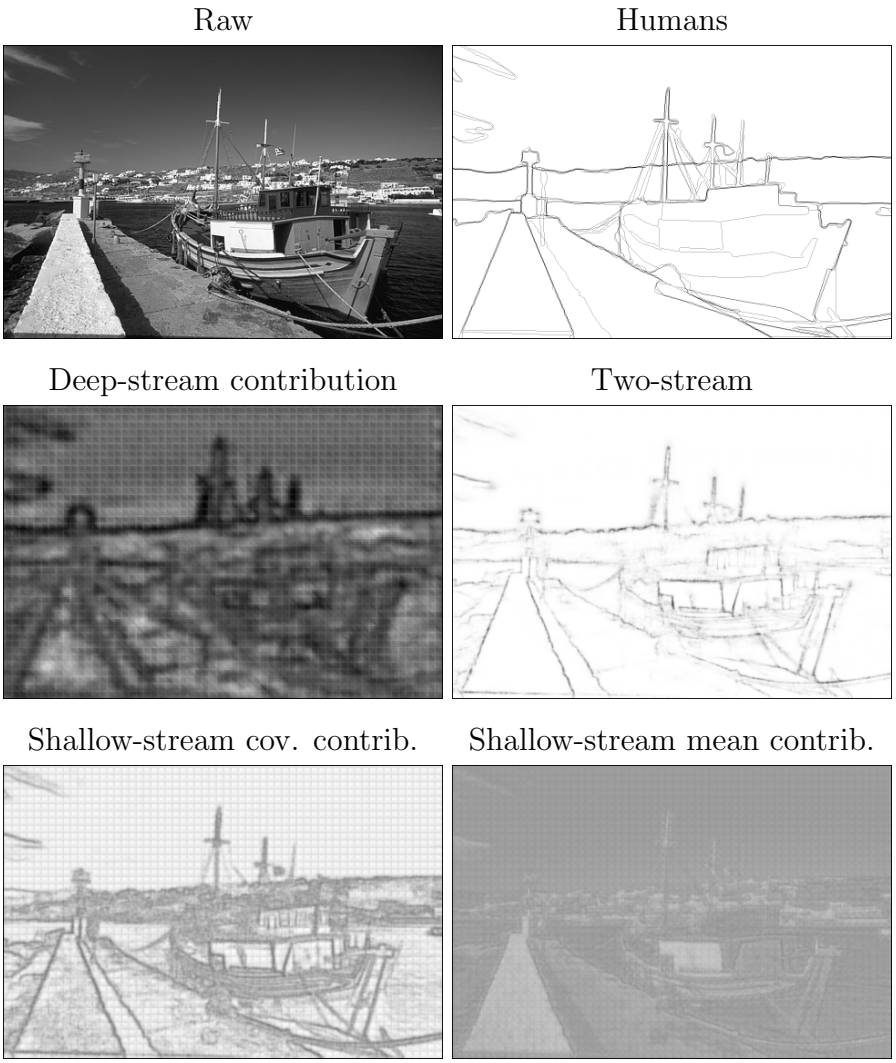


Figure D.10: Dissecting a contour prediction result on the grey-scale BSDS500. The stream contributions are normalized to be on the same scale, which also fills the full intensity range. Best viewed on screen.

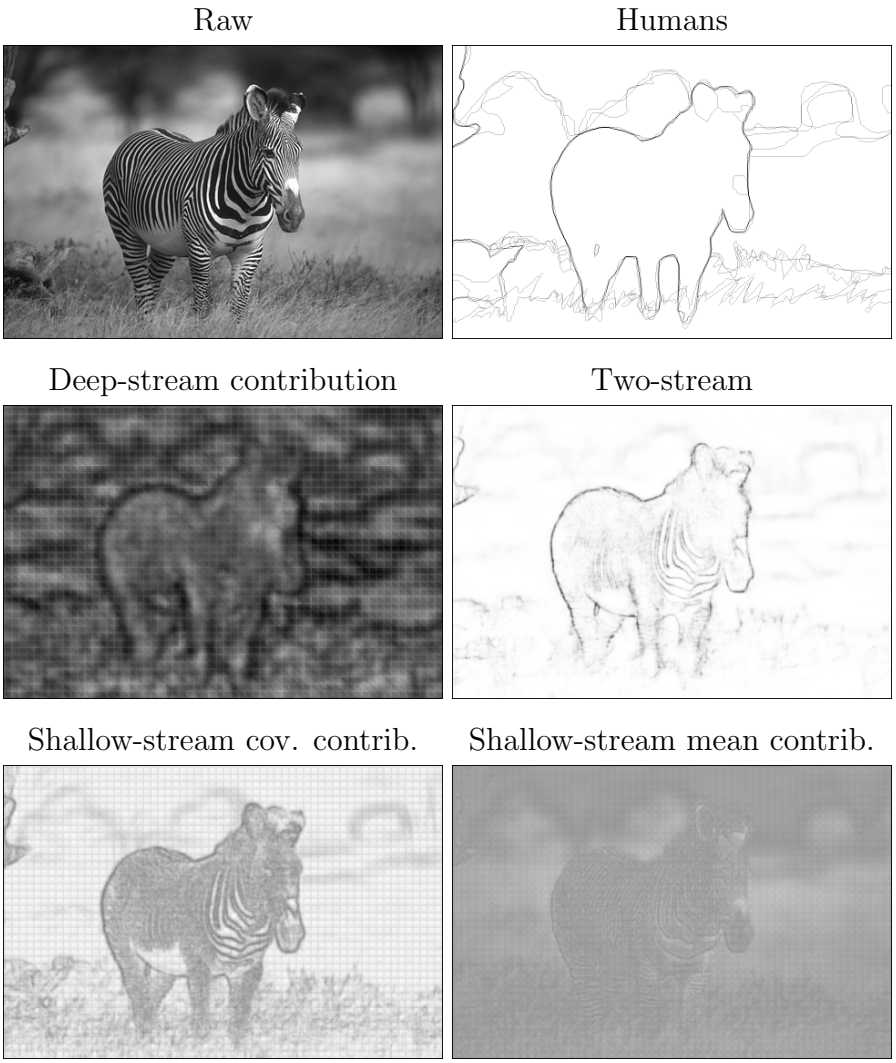


Figure D.11: Dissecting a contour prediction result on the grey-scale BSDS500. The stream contributions are normalized to be on the same scale, which also fills the full intensity range. Best viewed on screen.

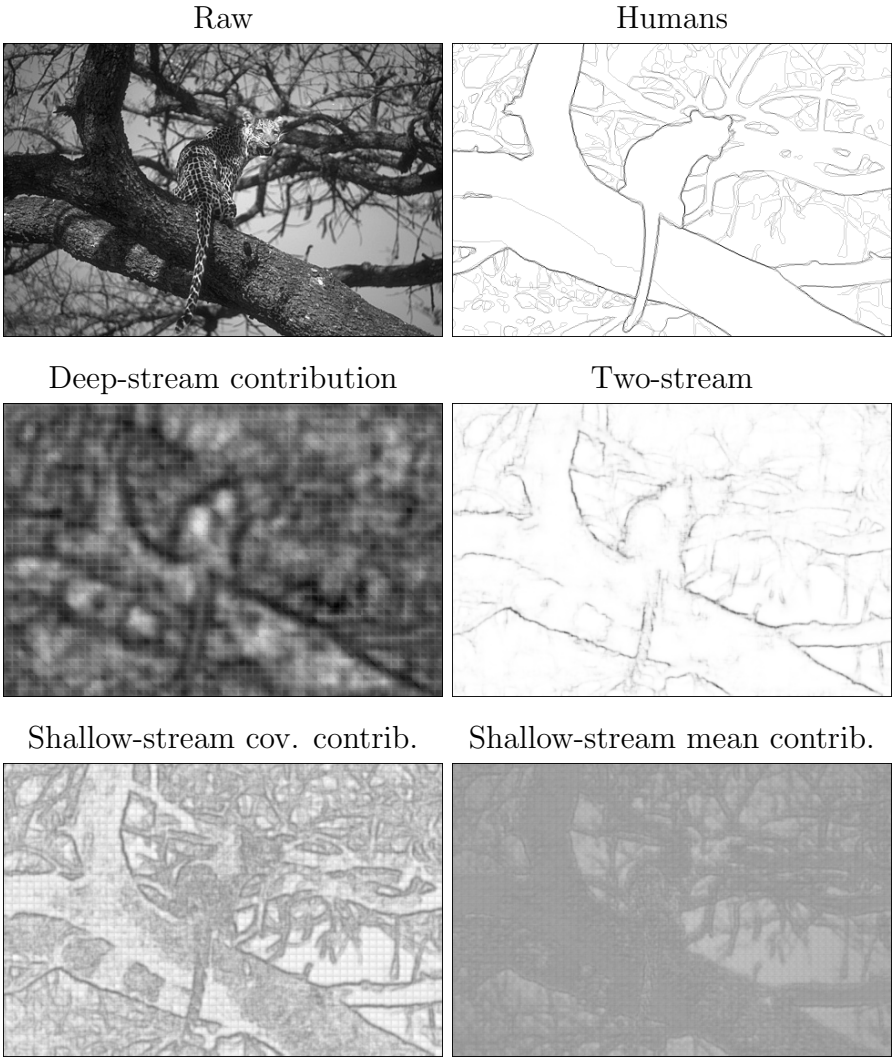


Figure D.12: Dissecting a contour prediction result on the grey-scale BSDS500. The stream contributions are normalized to be on the same scale, which also fills the full intensity range. Best viewed on screen.

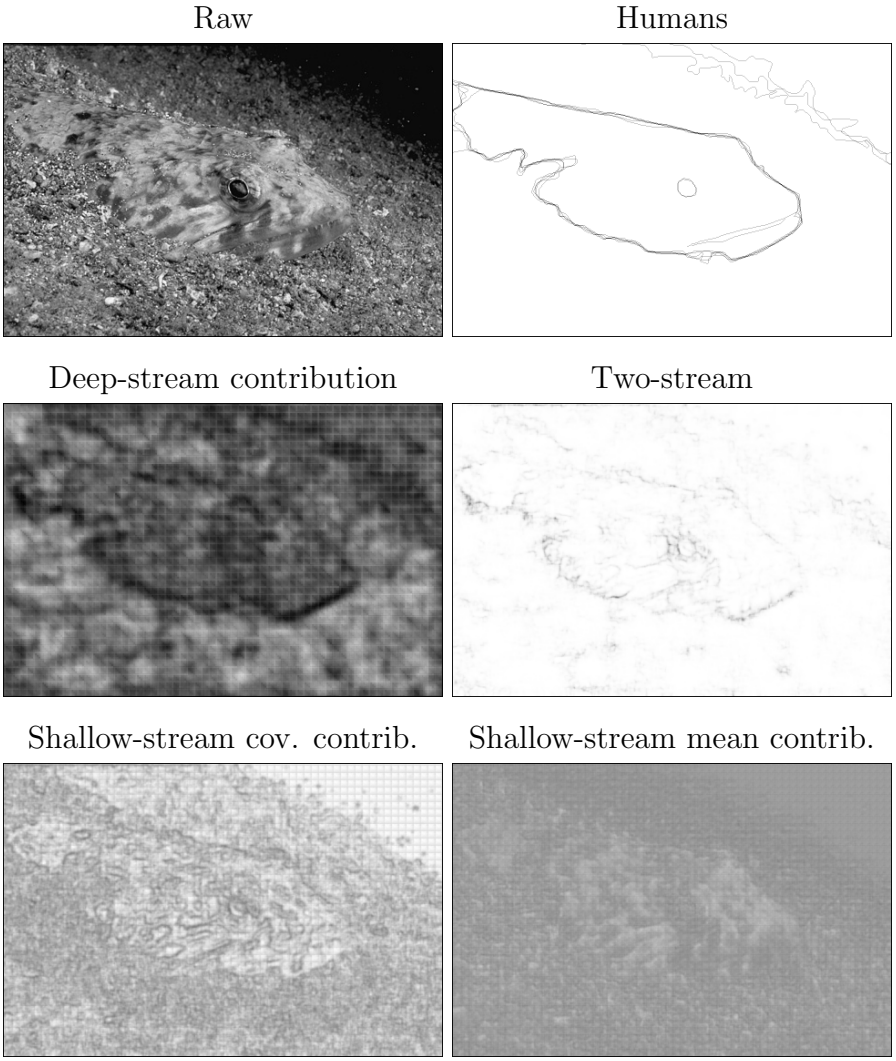


Figure D.13: Dissecting a contour prediction result on the grey-scale BSDS500. The stream contributions are normalized to be on the same scale, which also fills the full intensity range. Best viewed on screen.

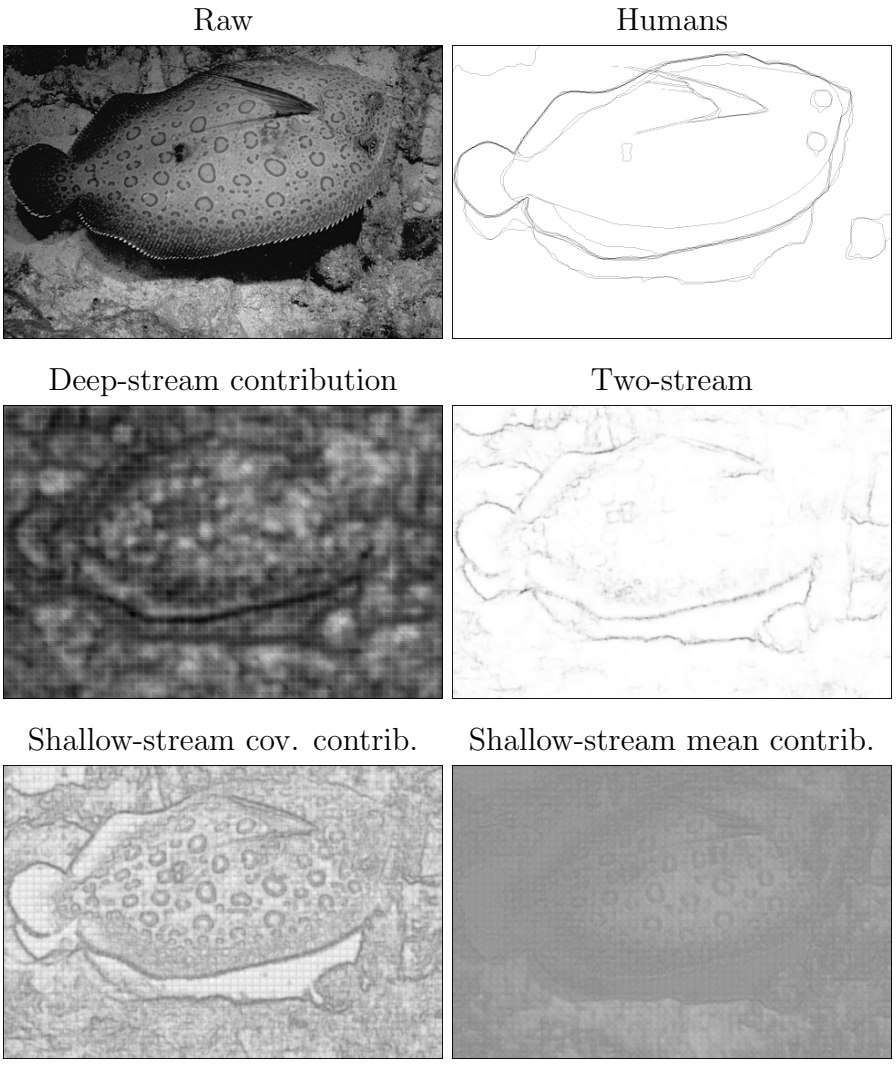


Figure D.14: Dissecting a contour prediction result on the grey-scale BSDS500. The stream contributions are normalized to be on the same scale, which also fills the full intensity range. Best viewed on screen.

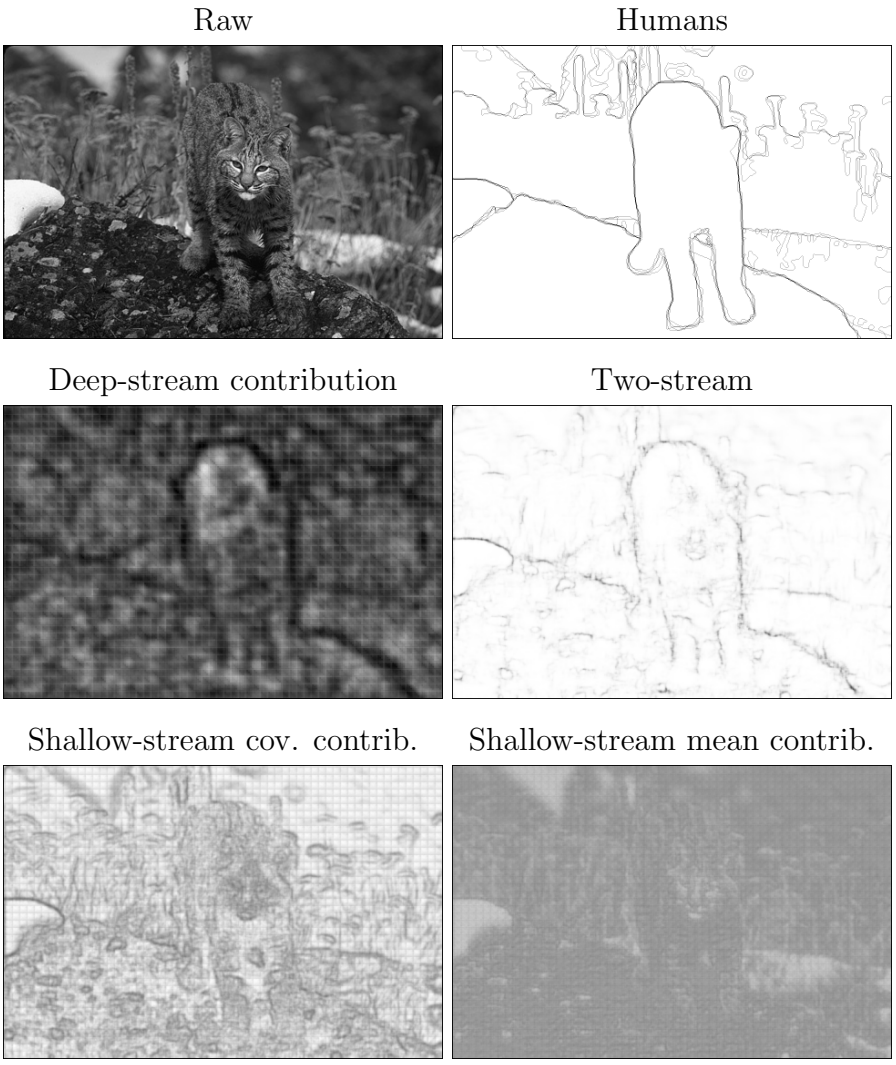


Figure D.15: Dissecting a contour prediction result on the grey-scale BSDS500. The stream contributions are normalized to be on the same scale, which also fills the full intensity range. Best viewed on screen.

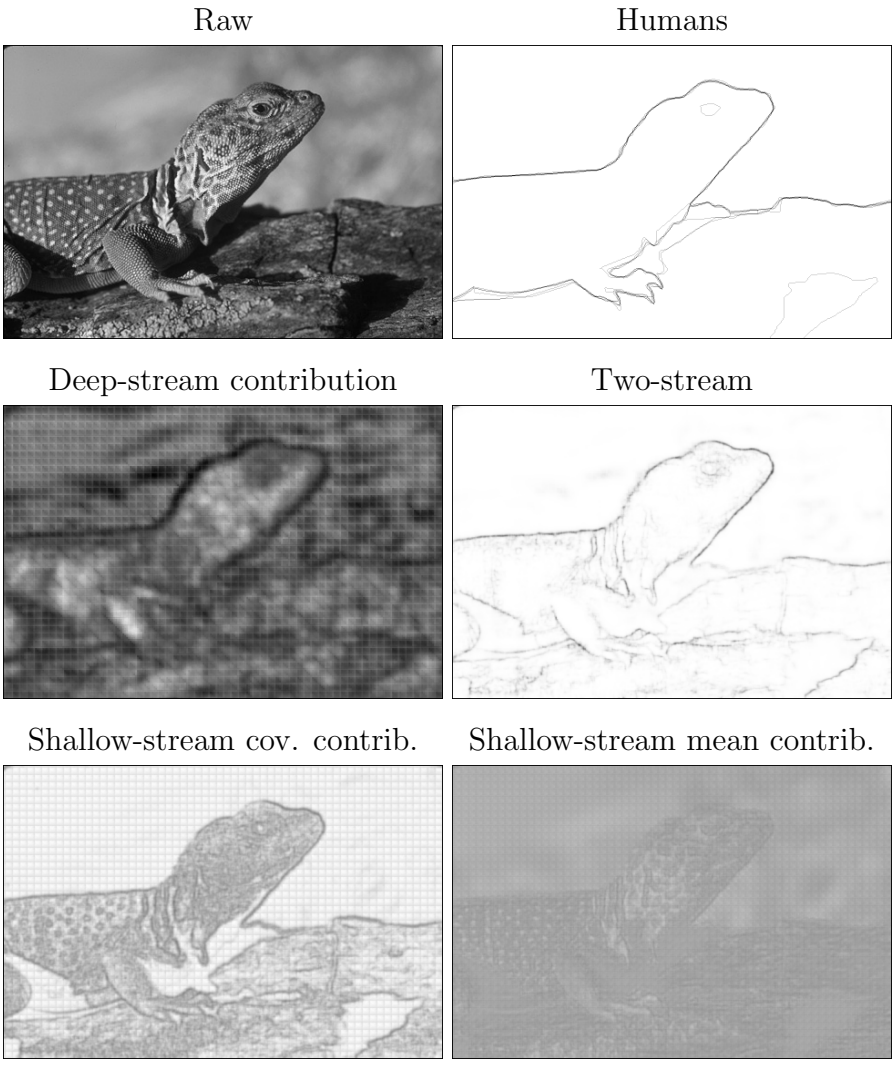


Figure D.16: Dissecting a contour prediction result on the grey-scale BSDS500. The stream contributions are normalized to be on the same scale, which also fills the full intensity range. Best viewed on screen.

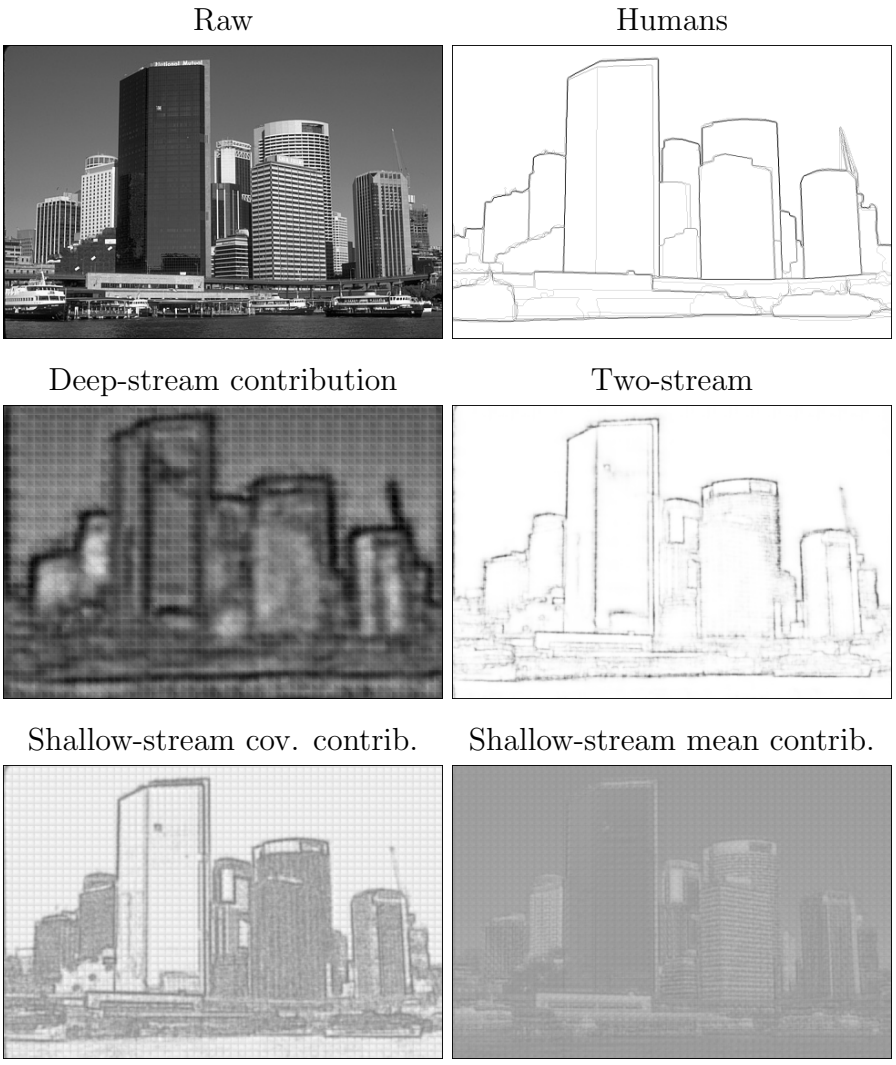


Figure D.17: Dissecting a contour prediction result on the grey-scale BSDS500. The stream contributions are normalized to be on the same scale, which also fills the full intensity range. Best viewed on screen.

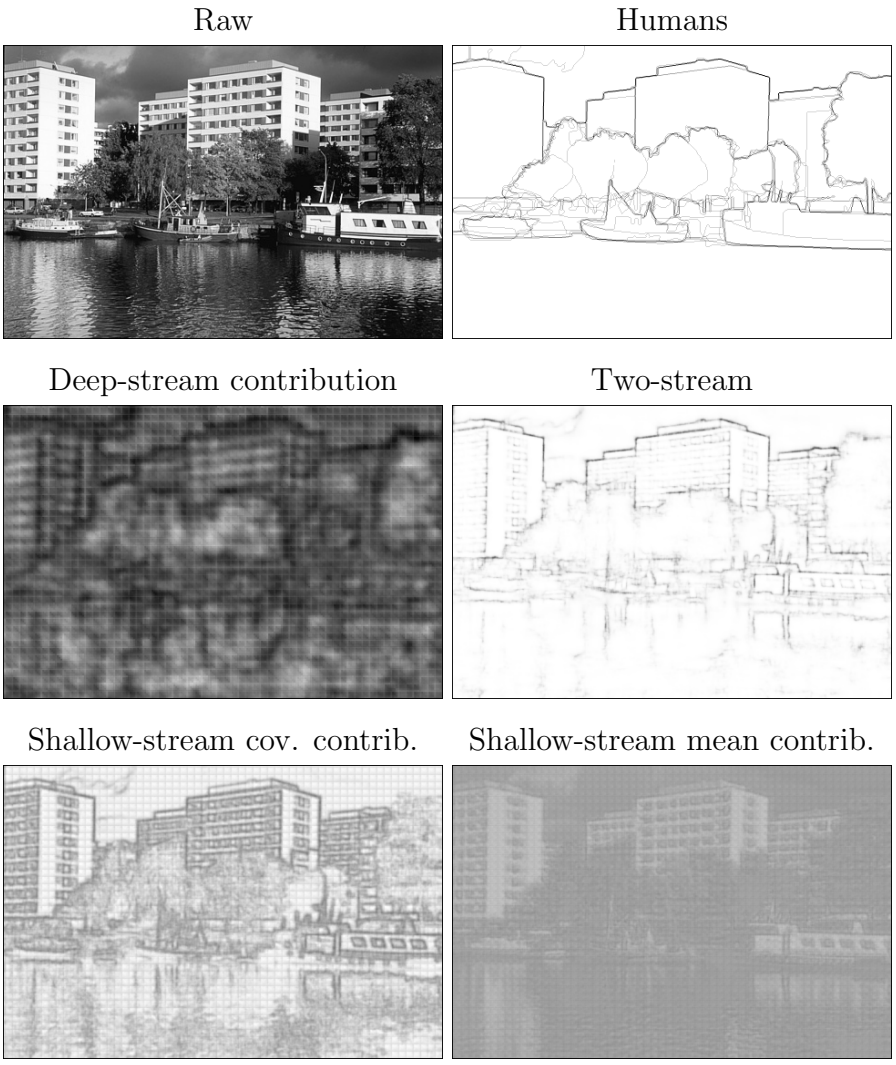


Figure D.18: Dissecting a contour prediction result on the grey-scale BSDS500. The stream contributions are normalized to be on the same scale, which also fills the full intensity range. Best viewed on screen.

units $\mathbf{v}$, and contour units $\mathbf{u}$) of the model is thus:

$$E(\mathbf{h},\mathbf{v},\mathbf{u}) = -\sum_j h_j^c \left(d_j - \frac{1}{2}\sum_f \pi_{fj} \left[\mathbf{K}_{\cdot f}{}^\top \mathbf{A}\mathbf{v}\right]^2 + \mathbf{W}_{\cdot j}^{c\top}\mathbf{u} \right) + \sum_i \frac{(v_i - a)^2}{2\sigma^2} - \sum_\ell h_\ell^m \left(b_\ell + \frac{1}{\sigma^2}\mathbf{M}_{\cdot \ell}{}^\top \mathbf{v} + \mathbf{W}_{\cdot \ell}^{m\top}\mathbf{u} \right) - g\sum_i u_i \quad \text{(D.19)}$$

where $\mathbf{v}$ denotes observed image pixels values, and $\mathbf{u}$ denotes observed contour unit states, whose grid sizes are the same. As in equation (5.1) we denote the filter of a factor unit with type f to image units as $\mathbf{K}_{\cdot f}$, and the filter of a mean hidden unit with index ℓ as $\mathbf{M}_{\cdot \ell}$. Similarly we denote the covariance hidden unit, mean hidden unit, and visible image biases by $\mathbf{d}$ $\mathbf{b}$, and a respectively, A is a whitening basis frontend, and σ is a positive scalar. The addition of contour units introduces three highlighted terms, in which $\mathbf{W}_{\cdot j}^c$ denotes covariance hidden unit with index j to to contour units filter, and $\mathbf{W}_{\cdot \ell}^m$ denotes mean hidden unit with index ℓ to contour units filter, and g a scalar contour unit bias. We denote these models as icRBMs, and their tiled-convolutional extensions as TicRBMs.

Figure D.19 illustrates an instance of a tiled-convolutional icRBM. The same technique can be used to merge information from the different modalities in the deeper layers, both defining instances of dual-wing harmoniums [Xing et al., 2005]. Such joint models would allow for also other kinds of interesting applications, including image-prediction (de-sketching) and image completion. For the latter task, boundary data might be available, and improve performance.

There are of course other ways of connecting the contour units to the hidden units, and also using different base-models. For example the contour units could be connected to PoT-style rather than cRBM-style covariance hidden units, however not without any additional constraints to retain a valid model: For valid models the distributions for a covariance hidden unit conditional on the visible units need to be Gamma-distributed with scale parameters greater than zero. Denoting θ_j as the scale parameter for a hidden unit with index j, we require that $\theta_j = 1 + \left[\mathbf{K}_{\cdot j}{}^\top \mathbf{A}\mathbf{v}\right]^2 + \mathbf{W}_{\cdot j}^{c\top}\mathbf{u} \geq 0$, which clearly is dependent on the last term. Using the proposed formulation there are no such problems, and the conditional distributions are Bernoulli-distributed as before, but with the contour

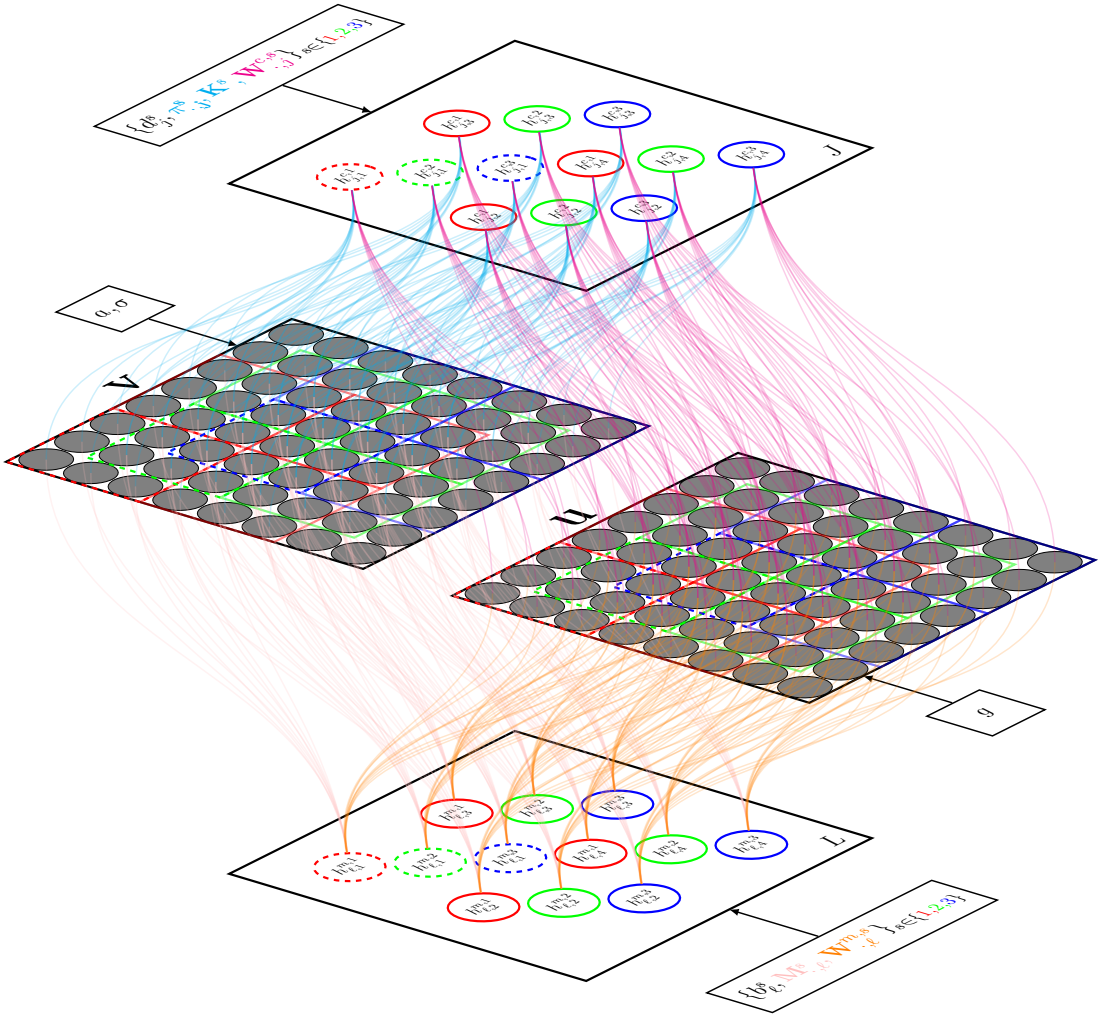


Figure D.19: A graphical illustration of a tiled-convolutional icRBM. The visible units (image brightness values $\mathbf{v}$ and contour states $\mathbf{u}$) visualized in the middle layers connect to the J covariance hidden unit layers of the model at the top, and to the L mean hidden unit layers of the model at the bottom. Within these layers hidden units are partitioned into different sets (red,green,blue), associated with different parameters for their hidden units. Each of the hidden units connect to a region of visible units (brightness values, contour states), and the filter applications within a set tile a two times the area horizontally and vertically, and those of different sets are offset diagonally with a stride of one unit between the neighboring sets, under both of the modalities of visible units. Note that even though all of the weight kernels specifying the connection weights between the hidden units of a certain type and the visible units have been drawn with uniform colours (cyan, magenta, pink, orange), they will typically have different values.

filter responses as additional terms:

$$\begin{aligned} p\big(h_j^c = 1 \mid \mathbf{v}, \mathbf{u}, d, \pi, \mathbf{K}, \mathbf{W}^c\big) &= \text{sig}\left(d_j - \frac{1}{2}\sum_f \pi_{fj}\left[\mathbf{K}_{\cdot f}{}^\top \mathbf{A}\mathbf{v}\right]^2 + \mathbf{W}_{\cdot j}^{c\,\top}\mathbf{u}\right), \\ p(h_\ell^m = 1 \mid \mathbf{v}, \mathbf{u}, a, \mathbf{M}, \mathbf{W}^m) &= \text{sig}\left(b_\ell + \frac{1}{\sigma^2}\mathbf{M}_{\cdot \ell}{}^\top \mathbf{v} + \mathbf{W}_{\cdot \ell}^{m\,\top}\mathbf{u}\right), \end{aligned} \tag{D.20}$$

where $\text{sig}(z) = 1/\exp(-z)$. The conditional distribution of contour units given hidden units is Bernoulli-distributed, with

$$p(u_i = 1 \mid \mathbf{h}, g, \mathbf{W}) = \text{sig}\left(g + \sum_j h_j^c W_{ij}^c + \sum_\ell h_\ell^m W_{i\ell}^m\right). \tag{D.21}$$

As the image and contour units are independent given the hidden units, the joint conditional distribution of visible units given hidden units is the same as in the regular mcRBM. As previously mentioned sampling such distribution is expected to be computationally demanding, as for large images large potentially full precision matrices would need to be computed (and inverted). However, it is possible to use a block-Gibbs sampling scheme, sampling first contour units conditional on hidden units as above, and image units conditional on the contour units running (say) a single step of HMC on the free-energy

$$\begin{aligned} F(\mathbf{v}; \mathbf{u}) = &-\sum_j \log\left\{1 + \exp\left\{d + \mathbf{W}_{\cdot j}^{c\,\top}\mathbf{u} - \frac{1}{2}\sum_f \pi_{fj}\left[\mathbf{K}_{\cdot f}{}^\top \mathbf{A}\mathbf{v}\right]^2\right\}\right\} \\ &+ \sum_i \frac{(v_i - a)^2}{2\sigma^2} - \sum_\ell \log\left\{1 + \exp\left\{b + \mathbf{W}_{\cdot \ell}^{m\,\top}\mathbf{u} + \frac{1}{\sigma^2}\mathbf{M}_{\cdot \ell}{}^\top \mathbf{v}\right\}\right\}, \end{aligned} \tag{D.22}$$

with step-size set automatically as done for learning TmcRBMs in previous section.

D.6.2 Partial derivatives in learning an icRBM

The free-energy under a tiled-convolutional icRBM (TicRBM) is the following:

$$\begin{aligned} F_{\text{TicRBM}}(\mathbf{v}, \mathbf{u}) = &-\sum_{s=1}^{S}\sum_{t=1}^{T}\sum_j \log\left\{1 + \exp\left\{d_j^s + \mathbf{C}_{\cdot j}^{s\,\top}\mathbf{u}_{\mathcal{N}_{s,t}} - \frac{1}{2}\sum_f \pi_{fj}\left[\mathbf{K}_{\cdot f}^{s\,\top}\mathbf{A}\mathbf{v}_{\mathcal{N}_{s,t}}\right]^2\right\}\right\} \\ &+ \sum_i \frac{(v_i - a)^2}{2\sigma^2} - \sum_{s=1}^{S}\sum_{t=1}^{T}\sum_\ell \log\left\{1 + \exp\left\{b_\ell^s + \mathbf{W}_{\cdot \ell}^{s\,\top}\mathbf{u}_{\mathcal{N}_{s,t}} + \frac{1}{\sigma^2}\mathbf{M}_{\cdot \ell}^{s\,\top}\mathbf{v}_{\mathcal{N}_{s,t}}\right\}\right\} - g\sum_i u_i, \end{aligned} \tag{D.23}$$

where $\mathbf{v}_{\mathcal{N}_{s,t}}$ denotes visible units in the tile with index t under a shift index s. There are S sets of parameters each containing J covariance unit biases, F factor filters, and a pooling matrix π^s of size $J \times F$. These are shared at T different locations, and assuming square images with $T^{1/2} \times T^{1/2}$ tile grid size, the image lattice is of size $T^{1/2}D + 2(D/2-1) \times T^{1/2}D + 2(D/2-1)$, where D is the even vertical and horizontal dimension of a square filter under the model. Partial derivatives of the free-energy with respect to the model parameters, and data are thus the following:

$$\frac{\partial F(\mathbf{v},\mathbf{u})}{\partial d^s_j} = -\sum_{t=1}^{T} \mathcal{S}(\mathrm{R}^{s,j}_t) \tag{D.24}$$

$$\frac{\partial F(\mathbf{v},\mathbf{u})}{\partial \mathbf{C}^s_{\cdot j}} = -\sum_{t=1}^{T} \mathcal{S}(\mathrm{R}^{s,j}_t)\mathbf{u}_{\mathcal{N}_{s,t}} \tag{D.25}$$

$$\frac{\partial F(\mathbf{v},\mathbf{u})}{\partial \mathbf{K}^s_{\cdot f}} = \sum_{t=1}^{T} \mathbf{A}\mathbf{v}_{\mathcal{N}_{s,t}} \sum_{j=1}^{J} \mathcal{S}(\mathrm{R}^{s,j}_t)\pi^s_{jf} \left[\mathbf{K}^s_{\cdot f}{}^{\top}\mathbf{A}\mathbf{v}_{\mathcal{N}_{s,t}}\right] \tag{D.26}$$

$$\frac{\partial F(\mathbf{v},\mathbf{u})}{\partial \pi^s_{jf}} = \frac{1}{2}\sum_{t=1}^{T} \mathcal{S}(\mathrm{R}^{s,j}_t)\mathbf{K}^s_{\cdot f}{}^{\top}\mathbf{A}\mathbf{v}_{\mathcal{N}_{s,t}} \tag{D.27}$$

$$\frac{\partial F(\mathbf{v},\mathbf{u})}{\partial b^s_\ell} = -\sum_{t=1}^{T} \mathcal{S}(\mathrm{Q}^{s,\ell}_t) \tag{D.28}$$

$$\frac{\partial F(\mathbf{v},\mathbf{u})}{\partial \mathbf{W}^s_{\cdot \ell}} = -\sum_{t=1}^{T} \mathcal{S}(\mathrm{Q}^{s,\ell}_t)\mathbf{u}_{\mathcal{N}_{s,t}} \tag{D.29}$$

$$\frac{\partial F(\mathbf{v},\mathbf{u})}{\partial \mathbf{M}^s_{\cdot \ell}} = -\sum_{t=1}^{T} \mathcal{S}(\mathrm{Q}^{s,\ell}_t)\mathbf{v}_{\mathcal{N}_{s,t}} \tag{D.30}$$

$$\frac{\partial F(\mathbf{v},\mathbf{u})}{\partial v_i} = \mathbf{A}\sum_{s,t:v_i\in\mathbf{v}_{\mathcal{N}_{s,t}}} \sum_{j=1}^{J} \mathcal{S}(\mathrm{R}^{s,j}_t) \sum_{f=1}^{F} \pi^s_{jf}\left[\mathbf{K}^s_{\cdot f}{}^{\top}\mathbf{A}\mathbf{v}_{\mathcal{N}_{s,t}}\right]\mathbf{K}^s_{s,t\to i,f} - \frac{1}{\sigma^2}\sum_{\ell} \mathcal{S}(\mathrm{Q}^{s,\ell}_t)\mathbf{M}^s_{s,t\to i,\ell},$$

where $\mathrm{R}^{s,j}_t = d^s_j + \mathbf{C}^s_{\cdot j}{}^{\top}\mathbf{u}_{\mathcal{N}_{s,t}} - \frac{1}{2}\sum_f \pi^s_{jf}\left[\mathbf{K}^s_{\cdot f}{}^{\top}\mathbf{A}\mathbf{v}_{\mathcal{N}_{s,t}}\right]^2$, $\mathrm{Q}^{s,\ell}_t = b^s_\ell + \mathbf{W}^s_{\cdot \ell}{}^{\top}\mathbf{u}_{\mathcal{N}_{s,t}} + \mathbf{M}^s_{\cdot \ell}{}^{\top}\mathbf{v}_{\mathcal{N}_{s,t}}$, $\mathcal{S}(x) = 1/(1+\exp\{-x\})$, the logistic function, and $\mathbf{K}^s_{t\to i,f}$,and $\mathbf{M}^s_{s,t\to i,\ell}$ denote the connection weights in $\mathbf{K}^s_{\cdot f}$ and in $\mathbf{M}^s_{\cdot \ell}$ associated with position s,t to visible unit v_i, respectively.

Bibliography

D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

N. J. Adams and C. K. I. Williams. Dynamic Trees for image modelling. *Image and Vision Computing*, 20(10):865–877, 2003.

C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50:5–43, 2003.

P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. PAMI*, 33(5):898–916, 2011.

A. Bell and T. J. Sejnowski. Edges are the 'independent components' of natural scenes. In *Proceedings, Advances in Neural Information Processing Systems (NIPS)*, pages 831–837. MIT Press, 1996.

Y. Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012.

Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Trans. PAMI*, 35(8):1798–1828, 2013.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995.

D. Blackwell. Conditional expectation and unbiased sequential estimation. *Annals of Mathematical Statistics*, 18(1):105–110, 1947.

L. Bottou. Stochastic gradient tricks. In G. Montavon, G. B. Orr, and K. Muller, editors, *Neural Networks, Tricks of the Trade, Reloaded*, Lecture Notes in Computer Science (LNCS 7700), pages 430–445. Springer, 2012.

S. Brooks and A. Gelman. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7:434–455, 1998.

J. Canny. A computational approach to edge detection. *IEEE Trans. PAMI*, 8(6):679–698, 1986.

B. Catanzaro, B. Y. Su, N. Sundaram, Y. Lee, M. Murphy, and K. Keutzer. Efficient, high-quality image contour detection. In *Proceedings, International Conference on Computer Vision (ICCV)*, pages 2381–2388, 2009.

K. Cho, A. Ilin, and T. Raiko. Improved learning of Gaussian-Bernoulli restricted Boltzmann machines. In *Proceedings, International Conference on Artificial Neural Networks (ICANN)*, 2011.

R. Coifman and D. Donoho. Translation invariant denoising. In A. Antoniadis and G. Oppenheim, editors, *Wavelets and Statistics*, Lecture Notes In Statistics, pages 125–150. Springer Verlag, 1995.

S. Darabi, E. Shechtman, C. Barnes, D. B. Goldman, and P. Sen. Image melding: combining inconsistent images using patch-based synthesis. *ACM Transactions on Graphics (SIGGRAPH)*, 31(4), 2012.

P. Dollar, Z. Tu, and S. Belongie. Supervised learning of edges and object boundaries. In *Proceedings, Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2006.

S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195(2):216 – 222, 1987.

A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *ACM Transactions on Graphics (SIGGRAPH)*, pages 341–346, 2001.

A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings, International Conference on Computer Vision (ICCV)*, pages 1033–1038. 1999.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision (IJCV)*, 88(2):303–338, 2010.

S. Fidler and A. Leonardis. Towards scalable representations of visual categories: Learning a hierarchy of parts. In *Proceedings, Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

Y. Freund and D. Haussler. Unsupervised learning of distributions of binary vectors using 2-layer networks. In *Proceedings, Advances in Neural Information Processing Systems (NIPS)*, volume 4, pages 912–919, 1992.

P. J. Garrigues and B. A. Olshausen. Learning horizontal connections in a sparse coding model of natural images. In *Proceedings, Advances in Neural Information Processing Systems (NIPS)*, 2008.

A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin, editors. *Bayesian Data Analysis*. Chapman & Hall / CRC Press, 2003.

S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. PAMI*, 6:721–741, 1984.

D. B. Grimes and R. P. N. Rao. Bilinear sparse coding for invariant vision. *Neural Computation*, 17(1):47–73, 2005.

D. K. Hammond and E. P. Simoncelli. Image modelling and denoising with orientation-adapted Gaussian scale mixtures. *IEEE Trans. IP*, 17(11):2089–2101, 2008.

W. K. Hastings. Monte Carlo sampling methods using Markov Chains and their applications. *Biometrika*, 57(1):97–109, 1970.

N. Heess, C. K. I. Williams, and G.E. Hinton. Learning generative texture models with extended Fields-of-Experts. In *Proceedings, British Machine Vision Conference (BMVC)*, 2009.

A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *ACM Transactions on Graphics (SIGGRAPH)*, pages 327–340, 2001.

G. E. Hinton. Training products of experts by minimizing Contrastive Divergence. *Neural Computation*, 14:1771–1800, 2002a.

G. E. Hinton. Learning to represent visual input. *Phil. Trans. R. Soc. B*, 365: 177–184, 2010a.

G. E. Hinton. A practical guide to training restricted Boltzmann machines. Technical Report UTML TR 2010-003, Department of Computer Science, University of Toronto, 2010b.

G. E. Hinton. Training Products of Experts by minimizing Contrastive Divergence. *Neural Computation*, 14:1771–1800, 2002b.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

G. E. Hinton and T. J. Sejnowski. Optimal perceptual inference. In *Proceedings, Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 448–453, 1983.

G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Comp.*, 18:1527–1554, 2006.

X. Hou, A. Yuille, and C. Koch. Boundary detection benchmarking: Beyond F-measures. In *Proceedings, Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

A. Hyvarinen and P. O. Hoyer. Emergence of phase and shift invariant features by decomposition of natural images into independent feature subspaces. *Neural Comp.*, 12(7):1705–1720, 2000.

A. Hyvarinen, J. Hurri, and P. O. Hoyer. *Natural Image Statistics*. Springer-Verlag, 2009.

Y. Jin and S. Geman. Context and hierarchy in a probabilistic image model. In *Proceedings, Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2145–2152, 2006.

Y. Karklin and M. S. Lewicki. Learning higher-order structures in natural images. *Network: Computation in Neural Systems*, 14:483–499, 2003.

Y. Karklin and M. S. Lewicki. A hierarchical Bayesian model for learning nonlinear statistical regularities in nonstationary natural signals. *Neural Comp.*, 17(2):397–424, 2005.

J. J. Kivinen and C. K. I. Williams. Transformation equivariant Boltzmann machines. In *Proceedings, International Conference on Artificial Neural Networks (ICANN)*, pages 1–8, 2011.

J. J. Kivinen and C. K. I. Williams. Multiple texture Boltzmann machines. In *Proceedings, International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

J. J. Kivinen, E. B. Sudderth, and M. I. Jordan. Image denoising with nonparametric hidden Markov trees. In *Proceedings, International Conference on Image Processing (ICIP)*, volume 3, pages 121–124, 2007a.

J. J. Kivinen, E. B. Sudderth, and M. I. Jordan. Learning multiscale representations of natural scenes using Dirichlet processes. In *Proceedings, International Conference on Computer Vision (ICCV)*, pages 1–8, 2007b.

J. J. Kivinen, E. B. Sudderth, and M. I. Jordan. Hierarchical Dirichlet process hidden Markov trees for multiscale image analysis, 2013a. In preparation.

J. J. Kivinen, C. K. I. Williams, and N. M. O. Heess. Multistream networks for visual boundary prediction, 2013b. In preparation.

I. Kokkinos. Boundary detection using F-measure-, filter-, and feature- (F^3) boost. In *Proceedings, European Conference on Computer Vision (ECCV)*, pages 650–663, 2010.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

U. Koster, J. T. Lindgren, and A. Hyvarinen. Estimating Markov random field potentials for natural images. In *Proceedings of the International Conference on Independent Component Analysis and Signal Separation*, 2009.

V. Kwatra, I. Essa, A. Bobick, and N. Kwatra. Texture optimization for example-based synthesis. *ACM Transactions on Graphics (SIGGRAPH)*, 2005.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

H. Lee, C. Ekanadham, and A. Y. Ng. Sparse deep belief net model for visual area V2. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Proceedings, Advances in Neural Information Processing Systems (NIPS)*, pages 873–880. MIT Press, 2008.

H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings, International Conference on Machine Learning (ICML)*. 2009.

V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D . Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. *SIGARCH Comput. Archit. News*, 38(3):451–460, June 2010.

J. Lim, L. Zitnick, and P. Dollar. Sketch tokens: A learned mid-level representation for contour and object detection. In *Proceedings, Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

Y. Liu, W.-C. Lin, and J. H. Hays. Near-regular texture analysis and manipulation. *ACM Transactions on Graphics (SIGGRAPH)*, 23(3), 2004.

Z. Q. Liu, C. Liu, H.Y. Shum, and Y. Z. Yu. Pattern-based texture metamorphosis. In *Pacific Conference on Computer Graphics and Applications*, pages 184–193, 2002.

D. G. Lowe. Distinctive image features from scale–invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004.

H. Luo, P. L. Carrier, A. Courville, and Y. Bengio. Texture modeling with convolutional Spike-and-Slab RBMs and deep extensions. In *Proceedings, International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 415–423, 2013.

S. Lyu and E. P. Simoncelli. Modeling multiscale subbands of photographic images with fields of Gaussian scale mixtures. *IEEE Trans. PAMI*, 31(4):693–706, Apr 2009.

D. J. C. MacKay. Introduction to Monte Carlo methods. In M. I. Jordan, editor, *Learning in Graphical Models*, NATO Science Series, pages 175–204. Kluwer Academic Press, 1998.

J. Mairal, M. Leordeanu, F. Bach, Hebert M., and Ponce J. Discriminative sparse image models for class-specific edge detection and image interpretation. In *Proceedings, European Conference on Computer Vision (ECCV)*, 2008.

S. G. Mallat. Multiresolution signal decomposition: The wavelet rerpresentation. *IEEE Trans. PAMI*, 11(7):674–693, 1989.

D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color and texture cues. *IEEE Trans. PAMI*, 26(5):530–549, May 2004.

D. Martin, C. Fowlkes, D. Tal, and J. Malik. Berkeley Segmentation Dataset and Benchmark: About the Benchmark. `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/`, 2013a. Page generated 8 Aug. 2013.

D. Martin, C. Fowlkes, D. Tal, and J. Malik. Berkeley Segmentation Dataset and Benchmark: Boundary Detection Benchmark: Algorithm Ranking. `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/bench/html/algorithms.html`, 2013b. Page generated 20 Feb. 2013.

W. Matusik, M. Zwicker, and F. Durand. Texture design using a simplicial complex of morphable textures. *ACM Transactions on Graphics (SIGGRAPH)*, 25(3), 2005.

R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order Boltzmann machines. Technical Report UTML TR 2009-003, Department of Computer Science, University of Toronto, 2009.

R. Memisevic and G. E. Hinton. Learning to represent spatial transformations with factored higher-order Boltzmann machines. *Neural Computation*, 22:1473–1492, 2010.

N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

M. C. Mozer, R. S. Zemel, M. Behrmann, and C. K. I. Williams. Learning to segment images using dynamic feature binding. *Neural Comp.*, 4(5):650–665, 1992.

V. Nair and G. E. Hinton. Implicit mixtures of restricted Boltzmann machines. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Proceedings, Advances in Neural Information Processing Systems (NIPS)*, pages 1145–1152. 2009.

V. Nair and G. E. Hinton. 3D object recognition using deep belief nets. In *Proceedings, Advances in Neural Information Processing Systems (NIPS)*. 2010a.

V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings, International Conference on Machine Learning (ICML)*, 2010b.

R. Neal. Connectionist learning of Belief Networks. *Artificial Intelligence*, 56: 71–113, 1992.

R. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, 1993.

R. M. Neal. MCMC using Hamiltonian dynamics. In S. Brooks, A. Gelman, G. Jones, and X.-L. Meng, editors, *Handbook of Markov Chain Monte Carlo*. Chapman & Hall / CRC Press, 2011.

R. M. Neal. *Bayesian Learning for Neural Networks*. Springer, New York, 1996. Lecture Notes in Statistics 118.

J. Ngiam, K. Pangwei, Z. Chen, S. Bhaskar, and A. Y. Ng. Sparse filtering. In *Proceedings, Advances in Neural Information Processing Systems (NIPS)*, 2011.

M. Norouzi, M. Ranjbar, and G. Mori. Stacks of convolutional restricted Boltzmann machines for shift-invariant feature learning. In *Proceedings, Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37:3311–3325, 1997.

B. A. Olshausen, C. Cadieu, J. Culpepper, and Warland D. K. Bilinear models of natural images. In *SPIE Proceedings vol. 6492: Human Vision Electronic Imaging XII*, 2007.

S. Osindero and G. E. Hinton. Modelling image patches with a directed hierarchy of Markov random fields. In *Proceedings, Advances in Neural Information Processing Systems (NIPS)*, 2008.

S. Osindero, M. Welling, and G. E. Hinton. Topographic product models applied to natural scene statistics. *Neural Computation*, 18(2):381–414, 2006.

P. Parent and S. W. Zucker. Trace inference, curvature consistency, and curve detection. *IEEE Trans. PAMI*, 11(8):823–839, 1989.

J. Portilla, V. Strela, M. J. Wainwright, and E. P. Simoncelli. Image denoising using scale mixtures of Gaussians in the wavelet domain. *IEEE Trans. IP*, 12 (11):1338–1351, 2003.

M. Prasad, A. Zisserman, A. W. Fitzgibbon, M. P. Kumar, and P. H. S. Torr. Learning class-specific edges for object detection and segmentation. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2006.

M. Ranzato and G. E. Hinton. Modeling pixel means and covariances using factorized third-order Boltzmann machines. In *Proceedings, Conference on Computer Vision and Pattern Recognition (CVPR)*. 2010.

M. Ranzato, A. Krizhevsky, and G. E. Hinton. Factored 3-way restricted Boltzmann machines for modeling natural images. In *Proceedings, International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2010a.

M. Ranzato, V. Mnih, and G. E. Hinton. Generating more realistic images using gated MRF's. In *Proceedings, Advances in Neural Information Processing Systems (NIPS)*. 2010b.

M. Ranzato, J. Susskind, V. Mnih, and G. E. Hinton. On deep generative models with applications to recognition. In *Proceedings, Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011a.

M. Ranzato, J. Susskind, V. Mnih, and G. E. Hinton. On deep generative models with applications to recognition. In *Proceedings, Conference on Computer Vision and Pattern Recognition (CVPR)*. 2011b.

X. Ren. Multi-scale improves boundary detection in natural images. In *Proceedings, European Conference on Computer Vision (ECCV)*, pages 533–545, 2008.

X. Ren and L. Bo. Discriminatively trained sparse code gradients for contour detection. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Proceedings, Advances in Neural Information Processing Systems (NIPS)*, pages 593–601, 2012.

B. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, UK, 1996.

S. Roth and M. J Black. Fields of experts: A framework for learning image priors. In *Proceedings, Conference on Computer Vision and Pattern Recognition (CVPR)*, pages II: 860–867, 2005.

S. Roth and M. J. Black. Steerable random fields. In *Proceedings, International Conference on Computer Vision (ICCV)*, 2007.

R. Ruiters, R. Schnabel, and R. Klein. Patch-based texture interpolation. In *Proceedings, Eurographics Symposium on Renderings 2010*. Eurographics Association, 2011.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

R. Salakhutdinov and G. Hinton. Deep Boltzmann machines. In *Proceedings, International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 5, pages 448–455, 2009.

U. Schmidt and S. Roth. Learning rotation-aware features: From invariant priors to equivariant descriptors. In *Proceedings, Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis.

E. P. Simoncelli. Statistical modeling of photographic images. In Alan Bovik, editor, *Handbook of Image and Video Processing*, chapter 4.7, pages 431–441. Academic Press, May 2005. 2nd edition.

E. P. Simoncelli and B. A. Olshausen. Natural image statistics and neural representation. *Annual Review of Neuroscience*, 24:1193–1216, May 2001.

E. P. Simoncelli, W. T. Freeman, E. H. Adelson, and D. J. Heeger. Shiftable multi-scale transforms. *IEEE Trans. IT*, 38(2):857–607, 1992.

P. Smolensky. Information processing in dynamical systems: foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. McGraw-Hill, New York, 1986.

K. Sohn and H. Lee. Learning invariant representations with local transformations. In *Proceedings, International Conference on Machine Learning (ICML)*, 2012.

A. Srivastava, A. B. Lee, E. P. Simoncelli, and S.-C. Zhu. On advances in statistical modeling of natural images. *Journal of Mathematical Imaging and Vision*, 18:17–33, 2003.

A. J. Storkey and C. K. I. Williams. Image modelling with Position-Encoding Dynamic Trees. *IEEE Trans. PAMI*, 25(7):859–871, 2003.

C. Sutton and A. McAllum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4), 2012.

R. Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer, 2010.

Y. W. Teh, M. Welling, S. Osindero, and G. E. Hinton. Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, 4:1235–1260, Dec 2003.

J. B. Tenenbaum and W. T. Freeman. Separating style and content with bilinear models. *Neural Computation*, 12(6):1247–1283, 2000.

T. Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings, International Conference on Machine Learning (ICML)*, pages 1064–1071, 2008.

T. Tieleman and G. E. Hinton. Using fast weights to improve Persistent Contrastive Divergence. In *Proceedings, International Conference on Machine Learning (ICML)*, pages 1033–1040. ACM New York, NY, USA, 2009.

A. Waibel, T. Hanazawa, G. E. Hinton, K. Shikano, and K. Lang. Phoneme recognition using Time Delay Neural Networks. *IEEE Trans. ASSP*, 37:328–339, 1989.

M. J. Wainwright, E. P. Simoncelli, and A. S. Willsky. Random cascades on wavelet trees and their use in analyzing and modeling natural images. *Applied and Computational Harmonic Analysis*, 11:89–123, 2001.

Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Perceptual image quality assessment: From error visibility to structural similarity. *IEEE Trans. IP*, 13(4):600–612, April 2004.

L.-Y. Wei, S. Lefebvre, V. Kwatra, and G. Turk. State of the art in example-based texture synthesis. In *Eurographics, State of the Art Report, EG-STAR*. Eurographics Association, 2009.

Y. Weiss and W. T. Freeman. What makes a good model of natural images ? In *Proceedings, Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

M. Welling, M. Rosen-Zvi, and G. E. Hinton. Exponential family harmoniums with an application to information retrieval. In *Proceedings, Advances in Neural Information Processing Systems (NIPS)*, volume 17, 2004.

A. S. Willsky. Multiresolution Markov models for signal and image processing. *Proc. IEEE*, 90(8):1396–1458, 2002.

E. P. Xing, R. Yan, and A. G. Hauptmann. Mining associated text and images with dual-wing harmoniums. In *Proceedings, Uncertainty in Artificial Intelligence (UAI)*, 2005.

L. Younes. Estimation and annealing for Gibbsian fields. *Henri Poincaré-Probabilités et Statistiques*, 24(2):269–294, 1988.

R. S. Zemel, C. K. I. Williams, and M. C. Mozer. Lending direction to neural networks. *Neural Networks*, 8(4):503–512, 1995.

J. Zhang, K. Zhou, L. Velho, B. Guo, and H.-Y. Shum. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Transactions on Graphics (SIGGRAPH)*, 22(3), 2003.

Y. Zhang and C. Sutton. Quasi-Newton Markov chain Monte Carlo. In *Proceedings, Advances in Neural Information Processing Systems (NIPS)*, 2011.

G. Zhu, Q. amd Song and J. Shi. Untangling cycles for contour grouping. In *Proceedings, International Conference on Computer Vision (ICCV)*, 2007.

L. Zhu, C. Lin, H. Huang, Y. Chen, and A. Yuille. Unsupervised structure learning: Hierarchical recursive composition, suspicious coincidence and competitive exclusion. In *Proceedings, European Conference on Computer Vision (ECCV)*, 2008.

S. C. Zhu and D. Mumford. A stochastic grammar of images. *Foundations and Trends in Computer Graphics and Vision*, 2(4):259–362, 2006.

S.-C. Zhu, Y. Wu, and D. Mumford. Filters, random fields and maximum entropy (FRAME): Towards a unified theory for texture modeling. *Int. J. Comput. Vision*, 27(2):107–126, 1998.