Deep Learning with Hierarchical Convolutional Factor Analysis

Bo Chen, *Member*, *IEEE*, Gungor Polatkan, Guillermo Sapiro, *Senior Member*, *IEEE*, David Blei, *Member*, *IEEE*, David Dunson, and Lawrence Carin, *Fellow*, *IEEE*

Abstract—Unsupervised multilayered ("deep") models are considered for imagery. The model is represented using a hierarchical convolutional factor-analysis construction, with sparse factor loadings and scores. The computation of layer-dependent model parameters is implemented within a Bayesian setting, employing a Gibbs sampler and variational Bayesian (VB) analysis that explicitly exploit the convolutional nature of the expansion. To address large-scale and streaming data, an online version of VB is also developed. The number of dictionary elements at each layer is inferred from the data, based on a beta-Bernoulli implementation of the Indian buffet process. Example results are presented for several image-processing applications, with comparisons to related models in the literature.

Index Terms—Bayesian, deep learning, convolutional, dictionary learning, factor analysis

1 Introduction

There has been significant recent interest in multilayered or "deep" models for representation of general data, with a particular focus on imagery and audio signals. These models are typically implemented in a hierarchical manner by first learning a data representation at one scale and using the factor scores or parameters learned at that scale as inputs for the next level in the hierarchy. Methods that have been considered include deconvolutional networks [1], convolutional networks [2], deep belief networks (DBNs) [3], hierarchies of sparse autoencoders [4], [5], [6], and convolutional restricted Boltzmann machines (RBMs) [7], [8], [9]. A key aspect of many of these algorithms is the exploitation of the convolution operator, which plays an important role in addressing large-scale problems, as one must typically consider all possible shifts of canonical bases or filters. In such analyses one must learn the form of the dictionary, as well as the associated coefficients. Concerning the latter, it has been recognized that a preference for sparse coefficients is desirable [1], [8], [9], [10].

When taking the outputs of layer l and using them as inputs for layer $l+1$, researchers have investigated several processing steps that improve computational efficiency and can also improve model performance. Specifically, researchers have pooled proximate parameters from layer l, and a processing step is employed before the coefficients are employed as inputs to layer $l+1$. It has been found desirable to just keep the maximum-amplitude coefficient within each pooled region; this is termed "max-pooling" [7], [8]. Max-pooling has two desirable effects: 1) As one moves to higher levels, the number of input coefficients diminishes, aiding computational speed; and 2) it has been found to improve classification [11], [12], [13].

Some of the multilayered models have close connections to overcomplete dictionary learning [14], in which image patches are expanded in terms of a sparse set of dictionary elements. The deconvolutional and convolutional networks in [1], [7], [9] similarly represent each level of the hierarchical model in terms of a sparse set of dictionary elements; however, rather than separately considering distinct patches as in [14], the work in [1], [7] allows all possible shifts of dictionary elements for representation of the entire image at once (not separate patches).

All of the methods discussed above, for "deep" models and for sparse dictionary learning for image patches, require one to specify a priori the number of dictionary elements employed within each layer of the model. In many applications, it may be desirable to infer the number of required dictionary elements based on the data itself. Within the deep models, for example, the dictionary-element coefficients at layer l are used (perhaps after pooling) as input features for layer $l+1$; this corresponds to a problem of inferring the proper number of features for the data of interest while allowing for all possible shifts of the dictionary elements, as in the various convolutional models discussed above. The idea of learning an appropriate number and composition of features has motivated the Indian buffet process (IBP) [15], as well as the beta-Bernoulli process to which it is closely connected [16], [17]. Such methods have been applied recently to (single-layer) dictionary learning in the context of image patches [18]. Further, the IBP has recently been employed for design of "deep" graphical models [19], although the problem considered in [19] did not consider multilayer feature learning.

- *B. Chen, G. Sapiro, and L. Carin are with the Electrical and Computer Engineering Department, Duke University, 130 Hudson Hall, Durham, NC 27708-0291. E-mail: lcarin@ee.duke.edu.*
- *G. Plotkan and D. Blei are with the Computer Science Department, Princeton University, 35 Olden St., Princeton, NJ 08540.*
- *D. Dunson is with the Statistics Department, Duke University, Box 90251, Durham, NC 27708-0251.*

Manuscript received 31 Mar. 2012; revised 28 Aug. 2012; accepted 16 Dec. 2012; published online 9 Jan. 2013.
Recommended for acceptance by S. Bengio, L. Deng, H. Larochelle, H. Lee, and R. Salakhutdinov.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number TPAMISI-2012-03-0236.
Digital Object Identifier no. 10.1109/TPAMI.2013.19.

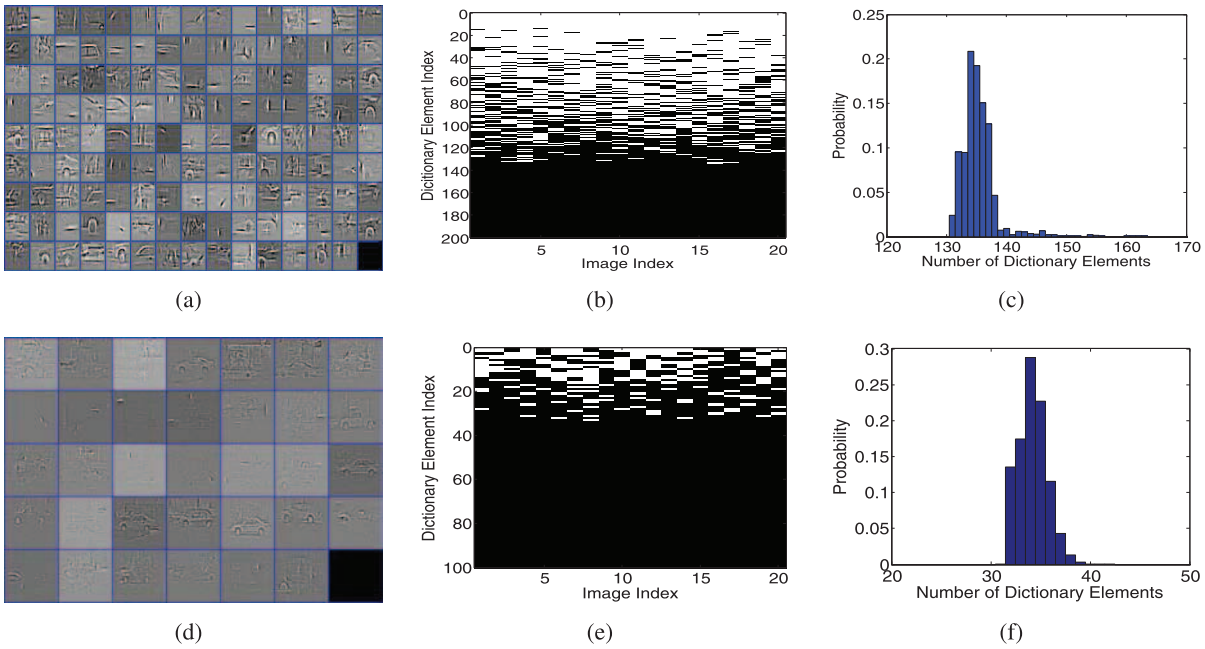


Fig. 1. Dictionary learned from the Car dataset. (a) Frequently used dictionary elements $\boldsymbol{d}_k^{(2)}$ at the second layer, organized left-to-right and top-down by their frequency of usage; (b) dictionary usage at layer 2, based on typical collection sample, for each of the images under analysis (white means used, black means not used); (c) approximate posterior distribution on the number of dictionary elements needed for layer 2, based upon Gibbs collection samples; (d) third-layer dictionary elements, $\boldsymbol{d}_k^{(3)}$; (e) usage of layer-3 dictionary elements for a typical Gibbs collection sample (white means used, black means not used); (f) approximate posterior distribution on the number of dictionary elements needed at layer 3. All dictionary elements $\boldsymbol{d}_k^{(2)}$ and $\boldsymbol{d}_k^{(3)}$ are shown in the image plane.

In this paper, we demonstrate that the idea of building an unsupervised deep model may be cast in terms of a hierarchy of factor-analysis models, with the factor scores from layer l serving as the input (potentially after pooling) to layer $l+1$. Of the various unsupervised deep models discussed above, the factor analysis view of hierarchical and unsupervised feature learning is most connected to [1], where an ℓ_1 sparseness constraint is imposed within a convolution-based dictionary expansion (equivalent to a sparseness constraint on the factor scores). In this paper, we consider four differences with previous deep unsupervised models:

1. the form of our model at each layer is different from that in [1] in that [1] performs optimization and here we employ Bayesian learning with different types of convolutional update equations;
2. the number of canonical dictionary elements or factor loadings at each layer is inferred from the data by an IBP/beta-Bernoulli construction;
3. sparseness on the dictionary-element coefficients (factor scores) at each layer is imposed with a Student-t prior (rather than ℓ_1 regularization);
4. fast computations are performed using Gibbs sampling and variational Bayesian (VB) analysis, where the convolution operation is exploited directly within the update equations.

Furthermore, to address large-scale datasets, including those arriving in a stream, online VB inference is also developed.

While the form of the proposed model is most related to [1], the characteristics of our results are more related to [7]. Specifically, when designing the model for particular image classes, the higher layer dictionary elements have a form that is often highly intuitive visually. To illustrate this and to also highlight unique contributions of the proposed model, consider Fig. 1, which will be described in further detail in Section 5; these results are computed with a Gibbs sampler, and similar results are found with batch and online VB analysis. In this example, we consider images from the "car" portion of the Caltech 101 image database. In Fig. 1a, we plot inferred dictionary elements at layer 2 in the model, and in Fig. 1d we do the same for dictionary elements at layer 3 (there are a total of three layers and the layer-1 dictionary elements are simple "primitives," as discussed in detail in Section 5). All possible shifts of these dictionary elements are efficiently employed, via convolution, at the respective layer in the model. Note that at layer 2 the dictionary elements often capture *localized* details on cars, while at layer 3 the dictionary elements often look like complete cars ("sketches" of cars). To the authors' knowledge, only the (very distinct) model in [7] achieved such physically interpretable dictionary elements at the higher layers in the model. A unique aspect of the proposed model is that we infer a posterior distribution on the required number of dictionary elements, based on Gibbs sampling, with the numerical histograms for these numbers shown in Figs. 1c and 1f for layers 2 and 3, respectively. Finally, the Gibbs sampler infers that dictionary elements are needed for expanding each layer, and in Figs. 1b and 1e the use of dictionary elements is shown for one Gibbs collection sample, highlighting the sparse expansion in terms of a small subset of the potential set of dictionary elements. These results give a sense of the characteristics of the proposed model, which is discussed in detail in the subsequent discussion.

The remainder of the paper is organized as follows: In Section 2, we develop the multilayer factor analysis viewpoint of deep models. The detailed form of the

Bayesian model is discussed in Section 3, and methods for Gibbs, VB, and online VB analysis are discussed in Section 4. A particular focus is placed on explaining how the convolution operation is exploited in the update equations of these iterative computational methods. Several example results are presented in Section 5, with conclusions provided in Section 6. An early version of the work presented here was published in a conference paper [20] which focused on a distinct *hierarchical* beta process construction, and in which the batch and online VB computational methods were not developed.

2 Multilayered Sparse Factor Analysis

2.1 Single Layer

The nth image to be analyzed is $\mathbf{X}_n \in \mathbb{R}^{n_y \times n_x \times K_c}$, where K_c is the number of color channels (e.g., for gray-scale images $K_c = 1$, while for RGB images $K_c = 3$). We consider N images $\{\mathbf{X}_n\}_{n=1,N}$, and each image $\mathbf{X}_n$ is expanded in terms of a dictionary, with the dictionary defined by compact canonical elements $\boldsymbol{d}_k \in \mathbb{R}^{n'_y \times n'_x \times K_c}$, with $n'_x \ll n_x$ and $n'_y \ll n_y$. The dictionary elements are designed to capture local structure within $\mathbf{X}_n$, and all possible two-dimensional (spatial) shifts of the dictionary elements are considered for representation of $\mathbf{X}_n$. For K canonical dictionary elements, the dictionary is $\{\boldsymbol{d}_k\}_{k=1,K}$. In practice, the number of dictionary elements K is made large, and we wish to infer the subset of dictionary elements actually needed to sparsely render $\mathbf{X}_n$ as

$$\mathbf{X}_n = \sum_{k=1}^{K} b_{nk} \mathbf{W}_{nk} * \boldsymbol{d}_k + \boldsymbol{\epsilon}_n, \tag{1}$$

where $*$ is the convolution operator and $b_{nk} \in \{0, 1\}$ indicates whether $\boldsymbol{d}_k$ is used to represent $\mathbf{X}_n$, and $\boldsymbol{\epsilon}_n \in \mathbb{R}^{n_y \times n_x \times K_c}$ represents the residual; note that each dictionary element $\boldsymbol{d}_k$ is available for all images n, while b_{nk} indicates whether it is utilized in the representation of image n. The matrix $\mathbf{W}_{nk}$ represents the factor score for factor loading (dictionary element) $\boldsymbol{d}_k$ for image $\mathbf{X}_n$, and the support of $\mathbf{W}_{nk}$ is $(n_y - n'_y + 1) \times (n_x - n'_x + 1)$, allowing for all possible shifts, as in a typical convolutional model [7]. Let $\{w_{nki}\}_{i \in \mathcal{S}}$ represent the components of $\mathbf{W}_{nk}$ where the set $\mathcal{S}$ contains all possible indexes for dictionary shifts. We impose within the model that most w_{nki} are sufficiently small to be discarded without significantly affecting the reconstruction of $\mathbf{X}_n$. A similar sparseness constraint was imposed in [1], [9], [10].

The construction in (1) may be viewed as a special class of factor models. Specifically, we can rewrite (1) as

$$\mathbf{X}_n = \sum_{k=1}^{K} b_{nk} \sum_{i \in \mathcal{S}} w_{nki} \boldsymbol{d}_{ki} + \boldsymbol{\epsilon}_n, \tag{2}$$

where $\boldsymbol{d}_{ki}$ represents a *shifted* version of $\boldsymbol{d}_k$, shifted such that the center of $\boldsymbol{d}_k$ is situated at $i \in \mathcal{S}$, and $\boldsymbol{d}_{ki}$ is zero padded outside the support of $\boldsymbol{d}_k$. The $\boldsymbol{d}_{ki}$ represent factor loadings, and as designed these loadings are sparse with respect to the support of an image $\mathbf{X}_n$ (because the $\boldsymbol{d}_{ki}$ have significant zero-padded extent). This is closely related to sparse factor analysis applied in gene analysis, in which the factor loadings are also sparse [21], [22]. Here, however, the factor loadings have a special structure: The sparse set of factor loadings $\{\boldsymbol{d}_{ki}\}_{i \in \mathcal{S}}$ correspond to the *same* zero-padded dictionary element $\boldsymbol{d}_k$, with the nonzero components shifted to all possible locations in the set $\mathcal{S}$. Details on the learning of model parameters is discussed in Section 3, after first developing the complete hierarchical model.

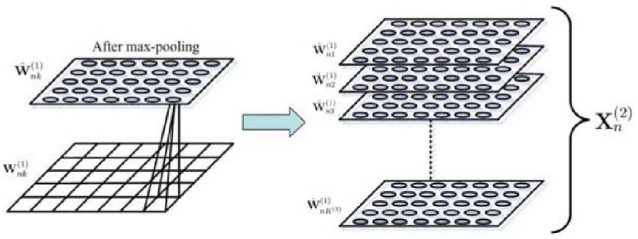


Fig. 2. Explanation of max pooling and collecting of coefficients from layer 1, for analysis at layer 2 (a similar procedure is implemented when transiting between any two consecutive layers in the hierarchical model). The matrix $\mathbf{W}^{(1)}_{nk}$ defines the shift-dependent coefficients $\{w^{(1)}_{nki}\}_{i \in \mathcal{S}^{(1)}}$ for all two-dimensional shifts of dictionary element $\{\boldsymbol{d}^{(1)}_{ki}\}_{i \in \mathcal{S}^{(1)}}$, for image n (this same max-pooling is performed for all images $n \in \{1, \ldots, N\}$). The matrix of these coefficients is partitioned with spatially contiguous blocks (bottom left). To perform max pooling, the maximum-amplitude coefficient within each block is retained and is used to define the matrix $\hat{\mathbf{W}}^{(1)}_{nk}$ (top-left). This max-pooling process is performed for all dictionary elements $\boldsymbol{d}^{(1)}_k$, $k \in \{1, \ldots, K^{(1)}\}$, and the set of max-pooled matrices $\{\hat{\mathbf{W}}^{(1)}_{nk}\}_{k=1,K^{(1)}}$ are stacked to define the tensor at right. The second-layer data for image n is $\mathbf{X}^{(2)}_n$, defined by a tensor like that at right. In practice, when performing stacking (right), we only retain the $\hat{K}^{(2)} \leq K^{(1)}$ layers for which at least one $b^{(1)}_{nki} \neq 0$, corresponding to those dictionary elements $\boldsymbol{d}^{(1)}_k$ used in the expansion of the N images.

2.2 Decimation and Max Pooling

For the nth image $\mathbf{X}_n$ and dictionary element $\boldsymbol{d}_k$, we have a set of coefficients (factor scores) $\{w_{nki}\}_{i \in \mathcal{S}}$, corresponding to all possible shifts in the set $\mathcal{S}$. A "max-pooling" step is applied to each $\mathbf{W}_{nk}$, with this employed previously in deep models [7] and in recent related image-processing analysis [11], [12]. In max pooling, each matrix $\mathbf{W}_{nk}$ is divided into a contiguous set of blocks (see Fig. 2), with each such block of size $n_{MP,y} \times n_{MP,x}$. The matrix $\mathbf{W}_{nk}$ is mapped to $\hat{\mathbf{W}}_{nk}$, with the mth value in $\hat{\mathbf{W}}_{nk}$ corresponding to the largest magnitude component of $\mathbf{W}_{nk}$ within the mth max-pooling region. Since $\mathbf{W}_{nk}$ is of size $(n_y - n'_y + 1) \times (n_x - n'_x + 1)$, each $\hat{\mathbf{W}}_{nk}$ is a matrix of size $(n_y - n'_y + 1)/n_{MP,y} \times (n_x - n'_x + 1)/n_{MP,x}$, assuming integer divisions.

To go to the second layer in the deep model, let $\hat{K}$ denote the number of dictionary elements $\boldsymbol{d}_k$ for which $b_{nk} \neq 0$ for at least one $n \in \{1, \ldots, N\}$. The $\hat{K}$ corresponding max-pooled images from $\{\hat{\mathbf{W}}_{nk}\}_{k=1,K}$ are stacked to constitute a datacube or tensor (see Fig. 2), with the tensor associated with image n now becoming the input image at the next level of the model. The max pooling and stacking is performed for all N images, and then the same form of factor modeling is applied to them (the original K_c color bands is now converted to $\hat{K}$ *effective* spectral bands at the next level). Model fitting at the second layer is performed analogously to that in (1).

After fitting the model at the second layer, to move to layer 3, max pooling is again performed, yielding a level-three tensor for each image, with which factor analysis is again performed. Note that, because of the max-pooling

step, the number of spatial positions in such images decreases as one moves to higher levels. Therefore, the basic computational complexity decreases with increasing layer within the hierarchy. This process may be continued for additional layers; in the experiments, we consider up to three layers.

2.3 Model Features and Visualization

Assume the hierarchical factor-analysis model discussed above is performed for L layers, and therefore, after max-pooling, the original image $\mathbf{X}_n$ is represented in terms of L tensors $\{\mathbf{X}_n^{(l)}\}_{l=2,L+1}$. The index l increases as one moves up the hierarchy away from the image plane, with $\hat{K}^{(l)}$ "spectral" bands at layer l, and $\mathbf{X}_n^{(1)}$ correspond to the original nth image, for which $\hat{K}^{(1)} = K_c$. It is of interest to examine the physical meaning of the associated dictionary elements (as shown in Fig. 1, for Layer-1 and Layer-2 dictionary elements).

A dictionary element at a layer $l > 1$ corresponds to a *set* of (generally contiguous) max-pooled factor scores from layer $l-1$. One may sequentially map a dictionary element from any layer $l > 1$ to a set of factor scores below, until at the lowest level the factor scores correspond to dictionary elements in the image plane. Because of the max-pool step, when performing such a synthesis, a coefficient at layer l must be associated with a location within the respective max-pool subregion at layer $l-1$. When synthesizing examples in Section 5 of dictionary elements projected onto the image plane, the coefficients are arbitrarily situated in the center of each max-pool subregion. This is only for visualization purposes, for illustration of the form of example dictionary elements; when analyzing a given image, the location of the maximum pixel within a max-pool subregion is not necessarily in the center.

3 Hierarchical Bayesian Analysis

We now consider an inference framework whereby we may perform the model fitting desired in (1), which is repeated at the multiple levels of the "deep" model. We wish to do this in a manner with which the number of required dictionary elements at each level may be inferred during the learning. Toward this end, we propose a Bayesian model based on an IBP [15] implementation of factor analysis, executed with a truncated beta-Bernoulli process [16], [17].

3.1 Hierarchical Model

We employ a model of the form

$$\begin{aligned}
\mathbf{X}_n &= \sum_{k=1}^{K} b_{nk}\mathbf{W}_{nk} * \boldsymbol{d}_k + \boldsymbol{\epsilon}_n,\\
\boldsymbol{\epsilon}_n &\sim \mathcal{N}\big(\mathbf{0}, \mathbf{I}_P\gamma_n^{-1}\big),\\
b_{nk} &\sim \text{Bernoulli}(\pi_k),\\
w_{nki} &\sim \mathcal{N}(0, 1/\alpha_{nki}),\\
\pi_k &\sim \text{Beta}(1/K, b),\\
d_{kj} &\sim \mathcal{N}(0, 1/\beta_j),\quad j \in \{1,\ldots,J\},\\
\gamma_n &\sim \text{Gamma}(c,d), \alpha_{nki} \sim \text{Gamma}(e,f), \beta_j \sim \text{Gamma}(g,h),
\end{aligned} \tag{3}$$

where J denotes the number of pixels in the dictionary elements $\boldsymbol{d}_k$, and $\boldsymbol{d}_{kj}$ is the jth component of $\boldsymbol{d}_k$. Since the same basic model is used at each layer of the hierarchy, in (3) we do not employ model-layer superscripts, for generality. The integer P denotes the number of pixels in $\mathbf{X}_n$, and $\mathbf{I}_P$ represents a $P \times P$ identity matrix. The hyperparameters (e, f) and (g, h) are set to favor large α_{nki} and β_j, thereby imposing that the set of w_{nki} will be compressible or approximately sparse, which was also found useful for the dictionary elements $\boldsymbol{d}_k$ (which yields dictionary elements that look like sparse "sketches" of images, as shown in Fig. 1). The priors on $\boldsymbol{d}_k$ and $\mathbf{W}_{nk}$ are also called *automatic relevance determination* (ARD) models [23].

The IBP representation may be used to infer the number of dictionary elements appropriate for representation of $\{\mathbf{X}_n\}_{n=1,N}$, while also inferring the dictionary composition. In practice, we truncate K and infer the subset of dictionary elements actually needed to represent the data. This procedure has been found to be computationally efficient in practice. One could alternatively directly employ the IBP construction [15] in which the number of dictionary elements is treated as unbounded in the analysis.

3.2 Computations

Recalling that $\mathcal{S}$ indexes the set of all possible shifts within the image of any dictionary element $\boldsymbol{d}_k$, an important aspect of this model construction is that, when implementing a Gibbs sampler or VB analysis, all factor scores $\{w_{nki}\}_{i\in\mathcal{S}}$ may be updated efficiently using the convolution operator (convolving $\boldsymbol{d}_k$ with $\mathbf{X}_n$), which may be implemented efficiently via the FFT. Therefore, while the model appears computationally expensive to implement, because all possible dictionary shifts $i \in \mathcal{S}$ must be considered, efficient implementations are possible; this is discussed in detail in Section 4, where both the Gibbs sampler and VB implementations are summarized.

3.3 Utilizing the Bayesian Outputs

The collection samples manifested by a Gibbs sampler yield an ensemble of models, and VB yields a factorized approximation to the posterior of all model parameters. When performing max pooling, moving from layer l to layer $l+1$, a single model must be selected, and here we have selected the maximum-likelihood (ML) sample among the collection samples, or the ML point from the VB analysis. In future research, it will be of interest to examine fuller exploitation of the Bayesian model. The main utility of the Bayesian formulation in the applications presented here is that it allows us to use the data to infer the number of dictionary elements, with related ideas applied previously in a distinct class of deep models [24] (that did not consider shifted dictionary elements).

3.4 Relationship to Previous Models

In (3), recall that upon marginalizing out the precisions α_{nki} we are imposing a Student-t prior on the factor scores w_{nki} [23]. Hence, with appropriate settings of hyperparameters (e, f), the Student-t imposes that the factor scores w_{nki} should be (nearly) sparse. This is closely connected to the model in [1], [8], [9], [10], in which an ℓ_1 regularization is imposed on w_{nki}, and a single (point) estimate is inferred on all model parameters. In [1], the authors also effectively imposed a Gaussian prior on $\boldsymbol{\epsilon}_n$, as we have in (3). Hence,

there are two principal differences between the proposed model and that in [1]: 1) Here, the beta-Bernoulli/IBP construction allows one to infer the number of dictionary elements (factor loadings) needed to represent the data at a given layer in the hierarchy, while in [1] this number is set; 2) we infer the model parameters using both Gibbs sampling and VB, which yield an ensemble of models at each layer rather than a point estimate. As discussed above, here the main advantage of difference 2 is as a means to achieve difference 1 because in most of our results we are not yet exploiting the full potential of the ensemble of solutions manifested by the approximate posterior. Other differences include that 1) sparseness is imposed on the filter coefficients and filters themselves, via a Bayesian generalization of the ℓ_1 regularizer; and 2) to handle massive data collections, including those arriving in a stream, we develop online VB.

4 Exploiting Convolution in Inference

In this section, we introduce two ways to infer the parameters in the multilayered model, Gibbs sampling and VB. An online version of VB inference is also developed to allow the model to scale.

4.1 Gibbs Sampler

We may implement the posterior computation by a Markov chain Monte Carlo (MCMC) method based on Gibbs sampling. Samples are constituted by iteratively drawing each random variable (model parameters and latent variables) from its conditional posterior distribution given the most recent values of all the other random variables. In the proposed model, all conditional distributions used to draw samples are analytic and relatively standard in Bayesian analysis. Therefore, for conciseness, all conditional update equations are presented in the supplemental material, which can be found in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2013.19.

4.2 VB Inference

For layer l of the model, in VB inference we seek a distribution $Q(\Theta^l;\Gamma^l)$ to approximate the exact posterior $p(\Theta^l|\boldsymbol{X}^l)$, where $\Theta^l \equiv \{\boldsymbol{b}^l, \boldsymbol{W}^l, \boldsymbol{d}^l, \boldsymbol{\pi}^l, \boldsymbol{\gamma}^l, \boldsymbol{\alpha}^l, \boldsymbol{\beta}^l\}$. Our objective is to optimize the parameters Γ^l in the approximation $Q(\Theta^l;\Gamma^l)$. Toward that end, consider the lower bound of the marginal log likelihood of the observed data:

$$\begin{aligned}\tilde{F}(\Gamma^l) &= \int d\Theta^l q(\Theta^l;\Gamma^l)\ln\frac{p(\boldsymbol{X}^l)p(\Theta^l|\boldsymbol{X}^l)}{q(\Theta^l;\Gamma^l)} \\ &= \ln p(\boldsymbol{X}^l) - \mathrm{KL}(q(\Theta^l;\Gamma^l)\|p(\Theta^l|\boldsymbol{X}^l)).\end{aligned} \tag{4}$$

Note that the term $p(\boldsymbol{X}^l)$ is a constant with respect to Γ^l, and therefore, $\tilde{F}(\Gamma^l)$ is maximized when the Kullback-Leibler divergence $\mathrm{KL}(q(\Theta^l;\Gamma^l)\|p(\Theta^l|\boldsymbol{X}^l))$ is minimized. However, we cannot explicitly compute the KL divergence because $p(\Theta^l|\boldsymbol{X}^l)$ is unknown. Fortunately, the numerator term in $\tilde{F}(\Gamma^l)$ may be computed because $p(\boldsymbol{X}^l)p(\Theta^l|\boldsymbol{X}^l) = p(\boldsymbol{X}^l|\Theta^l)p(\Theta^l)$, and the prior $p(\Theta^l)$ and likelihood function $p(\boldsymbol{X}^l|\Theta^l)$ are available. To make computation of $\tilde{F}(\Gamma^l)$ tractable, we assume $q(\Theta^l|\Gamma^l)$ has a factorized form $q(\Theta^l;\Gamma^l) = \Pi_m q_m(\Theta^l_m;\Gamma^l_m)$. With appropriate choice of q_m, the variational expression $\tilde{F}(\Gamma^l)$ may be evaluated analytically. The VB update equations are provided in the supplemental material, available online.

4.3 Online VB Analysis

Since the above VB requires a full pass through all the images each iteration, it can be slow to apply to very large datasets and it is not naturally suited to settings where new data are constantly arriving. Therefore, we develop an online variational inference for the multilayered convolutional factor analysis model, building upon a recent online implementation of latent Dirichlet allocation [25]. In an online variational inference, stochastic optimization is applied to the variational objective. We subsample the data (in this case, images), compute an approximation of the gradient based on the subsample, and follow that gradient with a decreasing step size. The key insight behind efficient online variational inference is that coordinate ascent updates applied in parallel precisely form the natural gradient of the variational objective function. Although the online VB converges much faster for large datasets, the practical algorithm is nearly as simple as the batch VB algorithm. The difference is only the update of global parameters, i.e., π_k and $\boldsymbol{d}_k$ in convolutional factor analysis. Suppose we randomly subsample one image each iteration and the total number of sampled images is D, the lower bound of image n is defined as

$$\begin{aligned}\tilde{F}^l_n = \mathbb{E}_q\big[&\log\big(p(\boldsymbol{X}^l_n|\boldsymbol{b}^l_n,\{\boldsymbol{d}^l_k\}_{k=1}^{K^l},\boldsymbol{W}^l_n,\gamma_n)p(\boldsymbol{b}^l_n|\boldsymbol{\pi}^l) \\ &p(\boldsymbol{W}^l_n|\boldsymbol{\alpha}^l_n)p(\boldsymbol{\alpha}^l_n|e,f)p(\gamma^l_n|c,d)\big)\big] \\ &+ H\big(q(\boldsymbol{b}^l_n)\big) + H\big(q(\boldsymbol{W}^l_n)\big) + H\big(q(\boldsymbol{\alpha}^l_n)\big) + H\big(q(\gamma^l_n)\big) \\ &+ \frac{1}{D}\big[\mathbb{E}_q\big[log\big(p(\boldsymbol{\pi}^l|a)p\big(\{\boldsymbol{d}^l_k\}_{k=1}^{K^l}|\boldsymbol{\beta}^l\big)p(\boldsymbol{\beta}^l|g,h)\big)\big] \\ &+ H(q(\boldsymbol{\pi}^l)) + H\big(q\big(\{\boldsymbol{d}^l_k\}_{k=1}^{K^l}\big)\big) + H(q(\boldsymbol{\beta}^l))\big],\end{aligned} \tag{5}$$

where $H(\cdot)$ is the entropy term for the variational distribution. Now, our goal is to maximize the lower bound

$$\tilde{F}^l = \sum_n \tilde{F}^l_n = \mathbb{E}_n\big[D\tilde{F}^l_n\big], \tag{6}$$

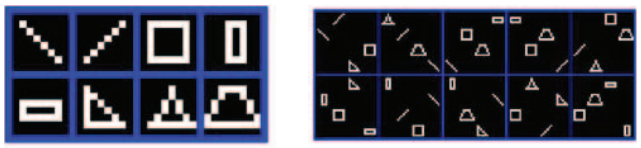
where the expectation is taken over the empirical distribution of the dataset. The expression $D\tilde{F}^l_n$ is the variational lower bound evaluated with D duplicate copies of image n. Then, take the gradient of global parameters $(\boldsymbol{\xi}^l, \boldsymbol{\Lambda}^l, \boldsymbol{\tau}^l_1, \boldsymbol{\tau}^l_2)$ of $D\tilde{F}^l_n$ as follows:

$$\partial\boldsymbol{\Lambda}^l_k(n) = \mathbf{1} \oslash \big(D\langle\gamma^l_n\rangle\langle b^l_{nk}\rangle\langle\|\mathbf{W}^l_{nk}\|_2^2\rangle + \langle\beta^l_k\rangle\big) \tag{7}$$

$$\begin{aligned}\partial\boldsymbol{\xi}^l_k(n) = D\boldsymbol{\Lambda}^l_k \odot \big(&\langle b^l_{nk}\rangle\langle\gamma^l_n\rangle\big(\mathbf{X}^l_{-n} * \langle\mathbf{W}^l_{nk}\rangle \\ &+ \langle\boldsymbol{d}^l_k\rangle\langle\|\mathbf{W}^l_{nk}\|_2^2\rangle\big)\big)\end{aligned} \tag{8}$$

$$\partial\tau^l_{k1}(n) = D\langle b^l_{nk}\rangle + \frac{1}{K^l} \tag{9}$$

$$\partial\tau^l_{k2}(n) = D + b - D\langle b^l_{nk}\rangle. \tag{10}$$



(a) (b)

Fig. 3. (a) The eight 8×8 images at the top are used as building blocks for generation of synthesized images. (b) Ten generated images are manifested by arbitrarily selecting four of the eight canonical elements and situating them arbitrarily. The images are 32×32, and in all cases black is zero and white is one (binary images).

The symbol $\odot$ is the element-wise product operator, $\oslash$ is the element-wise division operator, and $\langle \cdot \rangle$ represents expectation of the argument. Similarly to the natural gradient algorithm, an appropriate learning rate ρ_t is also needed to ensure the parameters converge to a stationary point in online inference. Then, the updates of $\boldsymbol{\xi}^l$,$\boldsymbol{\Lambda}^l$, $\boldsymbol{\tau}_1^l$, and $\boldsymbol{\tau}_2^l$ become

$$\begin{aligned} &\boldsymbol{\xi}_k^l \leftarrow (1-\rho_t)\boldsymbol{\xi}_k^l + \rho_t \boldsymbol{\xi}_k^l(n), \quad \boldsymbol{\Lambda}_k^l \leftarrow (1-\rho_t)\boldsymbol{\Lambda}_k^l + \rho_t \boldsymbol{\Lambda}_k^l(n) \\ &\tau_{k1}^l \leftarrow (1-\rho_t)\tau_{k1}^l + \rho_t \tau_{k1}^l(n), \quad \tau_{k2}^l \leftarrow (1-\rho_t)\tau_{k2}^l + \rho_t \tau_{k2}^l(n). \end{aligned} \tag{11}$$

In our experiments, we use $\rho_t = (\tau_0 + t)^{-\kappa}$, where $\kappa \in (0.5, 1]$ and $\tau_0 > 0$. The overall learning procedure is summarized in Algorithm 1, provided in the supplemental material, available online.

5 Experimental Results

5.1 Parameter Settings

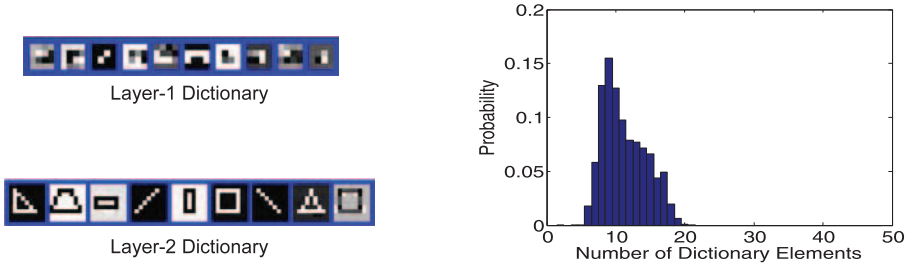
While the hierarchical Bayesian construction in (3) may appear relatively complex, the number of parameters that need to be set is not particularly large, and they are set in a "standard" way [17], [18], [23]. Specifically, for the beta-Bernoulli model $a = 1$ and $b = 1$, and for the gamma distributions $c = d = h = 10^{-6}$, $e = g = 1$, and $f = 10^{-3}$. The same parameters are used in all examples and in all layers of the "deep" model. The values of P, J, and K depend on the specific images under test (i.e., the image size), and these parameters are specified for the specific examples. In all examples we consider gray-scale images, and therefore $K_c = 1$. For online VB, we set the learning rate parameters as $\tau_0 = 1$ and $\kappa = 0.5$.

In the examples below, we consider various sizes for $\boldsymbol{d}_k^{(l)}$ at a given layer l, as well as different settings for the truncation levels $K^{(l)}$ and the max-pooling ratio. These parameters are selected as examples, and no tuning or optimization has been performed. Many related settings yield highly similar results, and the model was found to be robust to ("reasonable") variation in these parameters.

5.2 Synthesized and MNIST Examples

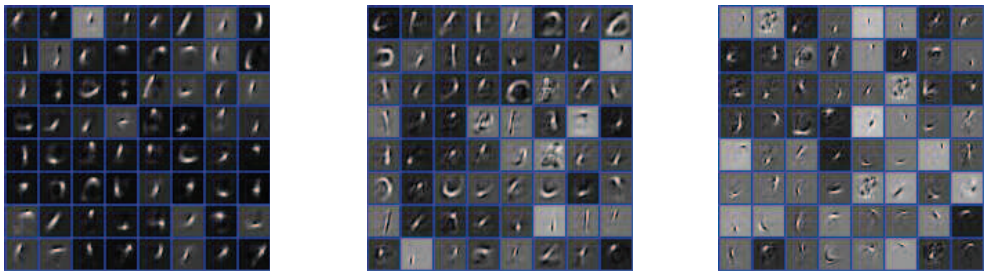
To demonstrate the characteristics of the model, we first consider synthesized data. In Fig. 3, we show eight canonical shapes, with shifted versions of these basic shapes used to constitute 10 example images (the latter are manifested in each case by selecting four of the eight canonical shapes, and situating them arbitrarily). The eight canonical shapes are binary and are of size 8×8; the 10 synthesized images are also binary, of size 32×32 (i.e., $P = 1,024$).

We consider a two-layer model, with the canonical dictionary elements $\boldsymbol{d}_k^{(l)}$ of size 4×4 ($J = 16$) at layer $l = 1$, and of spatial size 3×3 ($J = 9$) at layer $l = 2$. As demonstrated below, a two-layer model is sufficient to capture the (simple) structure in these synthesized images. In all examples below, we set the number of dictionary elements at layer 1 to a relatively small value, as at this layer the objective is to constitute simple primitives [3], [4], [5], [6], [7], [8]; here $K = 10$ at layer 1. For all higher level layers, we set K to a relatively large value, and allow the IBP construction to infer the number of dictionary elements needed; in this example $K = 100$ at layer 2. When performing max pooling, upon going from the output of layer 1 to the input of layer 2, and also when considering the output of layer 2, the down-sample ratio is two (i.e., the contiguous regions in which max pooling is performed are 2×2; see the left of Fig. 2). The dictionary elements inferred at layers 1 and 2 are depicted in Fig. 4; in both cases the dictionary elements are projected down to the image plane. Note that the dictionary elements $\boldsymbol{d}_k^{(1)}$ from layer 1 constitute basic elements, such as corners, horizontal, vertical, and diagonal segments. However, at layer 2 the dictionary elements $\boldsymbol{d}_k^{(2)}$, when viewed on the image plane, look like the fundamental shapes in Fig. 3a used to constitute the synthesized images. As an example of how the beta-Bernoulli distribution infers dictionary usage and number, from Fig. 4b we note that at layer 2 the posterior distribution on the number of dictionary elements is peaked



(a) (b)

Fig. 4. The inferred dictionary for the synthesized data in Fig. 3b. (a) Dictionary elements at layer 1, $\boldsymbol{d}_k^{(1)}$, and layer 2, $\boldsymbol{d}_k^{(2)}$; (b) estimated posterior distribution on the number of needed dictionary elements at level two, based upon the Gibbs collection samples. In (a) the images are viewed in the image plane. The layer 1 dictionary images are 4×4, and the layer 1 dictionary images are 9×9 (in the image plane, as shown). In all cases, white represents one and black represents zero.



(a) (b) (c)

Fig. 5. The inferred dictionary for MNIST data. (a) Layer-2 dictionary elements inferred by Gibbs sampler from 10,000 images; (b) Layer-2 dictionary elements inferred by batch VB from 10,000 images; (c) Layer-2 dictionary elements inferred by online VB from 60,000 images. In all cases, white represents one and black represents zero.

at 9, with the nine elements from the ML sample shown in Fig. 4a, while as discussed above, eight basic shapes were employed to design the toy images. In these examples, we employed Gibbs sampler with 30,000 burn-in iterations, and the histogram is based upon 20,000 collection samples. We ran this many samples because of the computational efficiency for this problem; good results are obtained with far fewer samples.

We next consider the widely studied MNIST data,[1] which has a total of 60,000 training and 10,000 testing images, each 28×28, for digits 0 through 9. We perform analysis on 10,000 randomly selected images for Gibbs and batch VB analysis. We also examine online VB inference on the *whole* training set; this an example of the utility of online-VB for scaling to large problem sizes.

A two-layer model is considered as these images are again relatively simple. In this analysis, the dictionary elements at layer 1, $\boldsymbol{d}_k^{(1)}$, are 7×7, while the second layer, $\boldsymbol{d}_k^{(2)}$, are 6×6. At layer 1 a max-pooling ratio of three is employed and $K = 24$, and at layer 2 a max-pooling ratio of two is used, and $K = 64$.

In Fig. 5, we present the dictionary elements $\boldsymbol{d}_k^{(2)}$ at Layer 2, as inferred by Gibbs, batch VB, and online VB, in each case viewed in the image plane. Layer-1 dictionary elements are basic, and they look similar for all three computational methods and are omitted for brevity. Additional (and similar) Layer-1 dictionary elements are presented below. We note at layer 2 the $\boldsymbol{d}_k^{(2)}$ take on forms characteristic of digits and parts of digits.

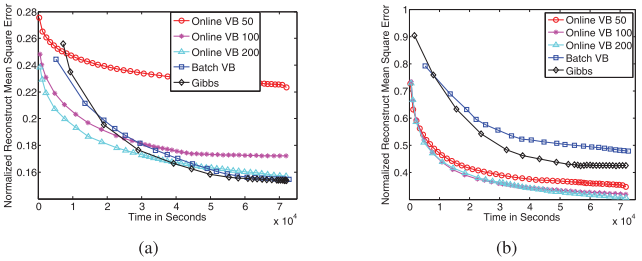
For the classification task, we construct feature vectors by concatenating the first and second (pooling) layer activations based on the 24 Layer 1 and 64 Layer-2 dictionary elements; results are considered based on Gibbs, batch VB, and online VB computations. The feature vectors are utilized within a nonlinear support vector machine (SVM) [26] with Gaussian kernel, in a one-versus-all multiclass classifier. The parameters in the SVM are tuned by fivefold cross validation. The Gibbs sampler obtained an error rate of 0.89 percent, online VB 0.96 percent, and batch VB 0.95 percent. The results from this experiment are summarized in Table 1, with comparison to several recent results on this problem. The results are competitive with the very best in the field, and are deemed encouraging, because they are based on features learned by a general purpose, unsupervised model.

1. http://yann.lecun.com/exdb/mnist/.

We now examine the computational costs of the different computational methods, relative to the quality of the model fit. As a metric, we use normalized reconstruct mean square error (RMSE) on held-out data, considering 200 test images each for digits 0 to 9 (2,000 test images in total). When performing computations on held-out data, the dictionary is fixed, and the factor scores are then inferred as done when performing initial learning (but without learning the dictionary). In Fig. 6, we plot the RMSE at Layers 1 and 2, as a function of computation time for batch VB, online VB (with different mini-batch sizes), and Gibbs sampling. For the batch VB and Gibbs solutions, all 10,000 *training* images are processed at once for model learning, and the

TABLE 1
Classification Performance for the MNIST Data Set of the Proposed Model (Denoted cFA, for Convolutional Factor Analysis) Using Three Inference Methods, with Comparisons to Approaches from the Literature

Methods	Test error
EmbedNN [27]	1.50%
DBN [3]	1.20%
CDBN [7]	0.82%
Ranzato *et al.*[28]	0.64%
Jarret *et al.* [4]	0.53%
cFA MCMC	0.89%
cFA Batch VB	0.95%
cFA online VB	0.96%



(a) (b)

Fig. 6. Held-out RMSE with different sizes of minibatches on MNIST data. Here, we use 10,000 images to train batch VB. For the Gibbs results, we consider the last sample up to a specified time, and we employ the same training data as in batch VB (online VB has the significant advantage of being able to handle much more data). (a) Layer 1, (b) layer 2.

computation time is directly linked to the number of VB iterations/Gibbs samples that are employed (all computations are on the same data). In the online VB solutions, batches of size 50, 100, or 200 are processed at a time (from the full 60,000 training images), with the images within a batch selected at random; for online VB the increased time reflected in Fig. 6 corresponds to the processing of more data, while increasing time for the batch (Gibbs and VB) results corresponds to more VB/Gibbs iterations. All computations were run on a desktop computer with Intel Core i7 920 2.26 GHz and 6-GB RAM, with the software written in Matlab. While none of the code has been carefully optimized, these results reflect relative computational costs, and relative fit performance.

Concerning Fig. 6, the largest batch size (200) yields the best model fit, and at very modest additional computational cost relative to smaller batches. At Layer 2, there is a substantial improvement in the model fit of the online VB methods relative to batch (recalling that the fit is measured on held-out data). We attribute that to the fact that online VB has the opportunity to sample (randomly) more of the training set, by the sequential sampling of different batches. The size of the batch training set is fixed and smaller, for computational reasons, and this apparently leads to Layer-2 dictionary elements that are well matched to the training data, but not as well matched and generalizable to held-out data. This difference between Layer 1 and Layer 2 relative batch versus online VB model fit is attributed to the fact that the Layer-2 dictionary elements capture more structural detail than the simple Layer-1 dictionary elements, and therefore, Layer-2 dictionary elements are more susceptible to overtraining.

5.3 Caltech 101 Data

The Caltech 101 dataset[2] is considered next. We rescale and zero-pad each image to 100×100, maintaining the aspect ratio, and then use local contrast normalization to preprocess the images (this is to be consistent with preprocessing steps considered in related models, e.g., [7]; this is not required, but it generally helps the visual form/structure of the inferred dictionary elements, with less impact on classification performance). Layer-1 dictionary elements $\boldsymbol{d}_k^{(1)}$ are of size 11×11, and the max-pooling ratio is 5. We consider 4×4 dictionary elements $\boldsymbol{d}_k^{(2)}$ and 6×6 dictionary elements $\boldsymbol{d}_k^{(3)}$. The max-pooling ratio at Layers 2 and 3 is set as 2. The beta-Bernoulli truncation level was set as $K = 200$. We first consider modeling each class of images separately, with a three-level model considered, as in [7]; because all of the images within a given class are similar, interesting and distinctive structure is manifested in the factor loadings, up to the third layer, as discussed below. In these experiments, the first set of results are based upon Gibbs sampling.

There are 102 image classes in the Caltech 101 dataset, and for conciseness we present results here for one (typical) class, revisiting Fig. 1; we then provide a summary exposition on several other image classes. The Layer-1 dictionary elements are depicted in Fig. 7, and we only focus on $\boldsymbol{d}_k^{(2)}$ and $\boldsymbol{d}_k^{(3)}$, from layers 2 and 3, respectively.

2. http://www.vision.caltech.edu/Image_Datasets/Caltech101/.

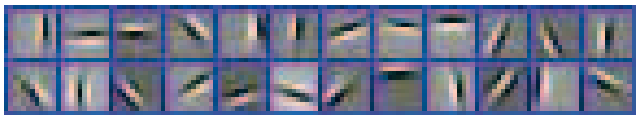


Fig. 7. Layer-1 dictionary elements learned by a Gibbs sampler for the Caltech 101 dataset.

Considering the $\boldsymbol{d}_k^{(2)}$ (in Fig. 1a), one observes several parts of cars, and for $\boldsymbol{d}_k^{(3)}$ (Fig. 1d) cars are often clearly visible. It is also of interest to examine the binary variable $b_{nk}^{(2)}$, which defines which of the candidate dictionary elements $\boldsymbol{d}_k^{(2)}$ are used to represent a given image. In Fig. 1b, we present the usage of the $\boldsymbol{d}_k^{(2)}$ (white indicates being used and black not used, and the dictionary elements are organized from most probable to least probable to be used). From Figs. 1c and 1f, one notes that of the 200 candidate dictionary elements, roughly 134 of them are used frequently at layer 2, and 34 are frequently used at layer 3. The Layer-1 dictionary elements for these data (typical of all Layer-1 dictionary elements for natural images) are depicted in Fig. 7.

In Fig. 8, we show Layer-2 and Layer-3 dictionary elements from five additional classes of the Caltech 101 dataset (the data from each class are analyzed separately). Note that these dictionary elements take on a form well matched to the associated image class. Similar class-specific Layer 2 and object-specific Layer-3 dictionary elements were found when each of the Caltech 101 data classes was analyzed in isolation. Finally, Fig. 9 shows Layer 2- and Layer 3-dictionary elements, viewed in the image plane, when the model trains *simultaneously* on five images from each of four classes (20 total images), where the four classes are faces, cars, planes, and motorcycle. We see in Fig. 9 that the Layer-2 dictionary elements seem to capture general characteristics of all of the classes, while at Layer 3 one observes specialized dictionary elements, specific to particular classes.

The total Caltech 101 dataset contains 9,144 images, and we next consider online VB analysis on these images; batch VB and Gibbs sampling is applied to a subset of these data. A held-out dataset of 510 images is selected at random, and we use these images to test the quality of the learned model to fit new data, as viewed by the model fit at Layers 1 and 2. Batch VB/Gibbs is trained on 1,020 images (the largest number for which reasonable computational cost could be achieved on the available computer), and online VB was considered using minibatch sizes of 10, 20, and 50. Fig. 11 demonstrates model fit to the held-out data, for the batch VB/Gibbs and online VB analysis, as a function of computation, in the same manner as performed in Fig. 6. For this case, the limitations of batch VB and Gibbs sampling (not able to use as much training data, and hence poorer generalization to held-out data) are evident at both Layers 1 and 2, which is attributed to the fact that the Caltech 101 data are more complicated than the MNIST data.

To examine dictionary-element usage, Fig. 12 we show Layer 2 dictionary usage for a set of 50 face test images. Note from Fig. 12b that for a given image typically more than half of the dictionary elements at this layer are not employed.

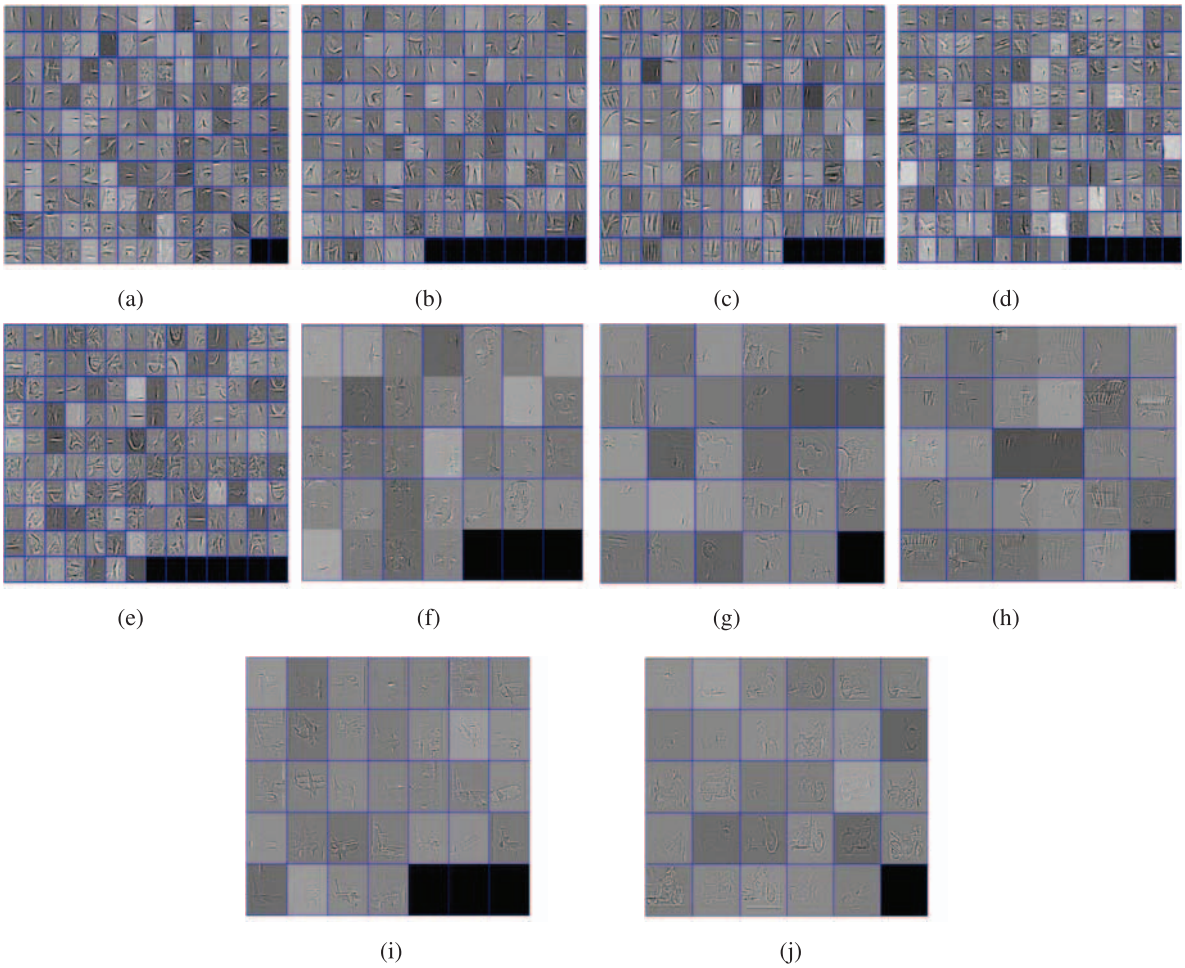


Fig. 8. Analysis of five datasets from Caltech 101, based upon Gibbs sampling. (a)-(e) Inferred Layer-2 dictionary elements, $\boldsymbol{d}_k^{(2)}$, respectively, for the following datasets: face, elephant, chair, airplane, motorcycle; (f)-(j) inferred Layer-3 dictionary elements, $\boldsymbol{d}_k^{(3)}$, for the same respective datasets. All of images are viewed in the image plane, and each of the classes of images were analyzed separately.

5.4 Layer-Dependent Activation

It is of interest to examine the factor scores $w_{nki}^{(l)}$ at each of the levels of the hierarchy, here for $l = 1, 2, 3$. In Fig. 13, we show one example from the face dataset, using Gibbs sampling and depicting the ML collection sample. Note that the set of factor scores becomes increasingly sparse with increasing model layer l, underscoring that at layer $l = 1$ the dictionary elements are fundamental (primitive), while with increasing l the dictionary elements become more specialized and, therefore, sparsely utilized. The corresponding reconstructions of the image in Fig. 13, as manifested via the three layers, are depicted in Fig. 14 (in all cases viewed in the image plane).

5.5 Sparseness

The role of sparseness in dictionary learning has been discussed in recent papers, especially in the context of structured or part-based dictionary learning [29], [30]. In deep networks, the ℓ_1-penalty parameter has been utilized

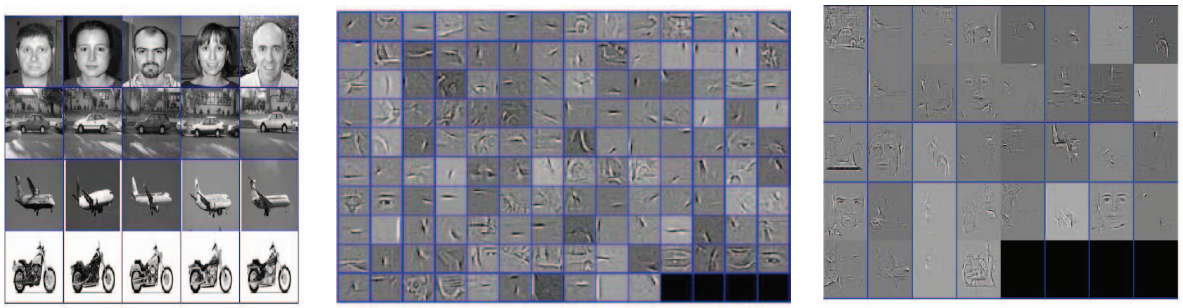
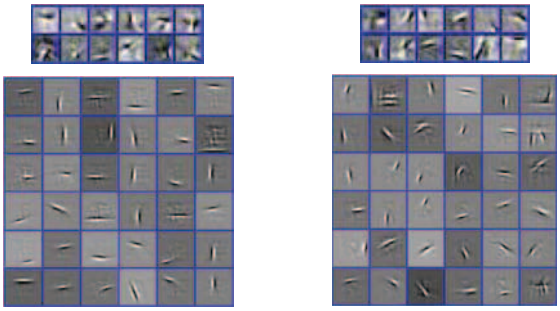
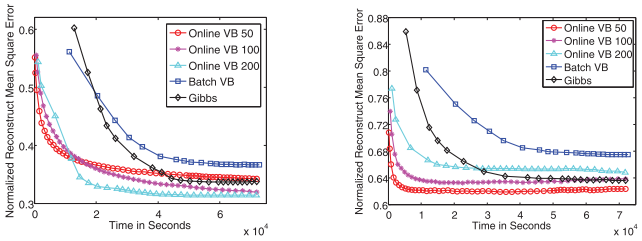


Fig. 9. Joint analysis of four image classes from Caltech 101, based on Gibbs sampling. (a) Original images; (b) Layer-2 dictionary elements $\boldsymbol{d}_k^{(2)}$, (c) Layer-3 dictionary elements $\boldsymbol{d}_k^{(3)}$. All figures in (b) and (c) are shown in the image plane.



(a) (b)

Fig. 10. The inferred Layer 1 and Layer 2 dictionary for Caltech 101 data. (a) Layer-1 and Layer-2 dictionary elements inferred by batch VB on 1,020 images; (b) Layer-1 and Layer-2 dictionary elements inferred by online VB on the whole dataset (9,144 images).



(a) (b)

Fig. 11. Held-out RMSE with different sizes of minibatches on Caltech101 data, as in Fig. 6. (a) Layer 1, (b) Layer 2.

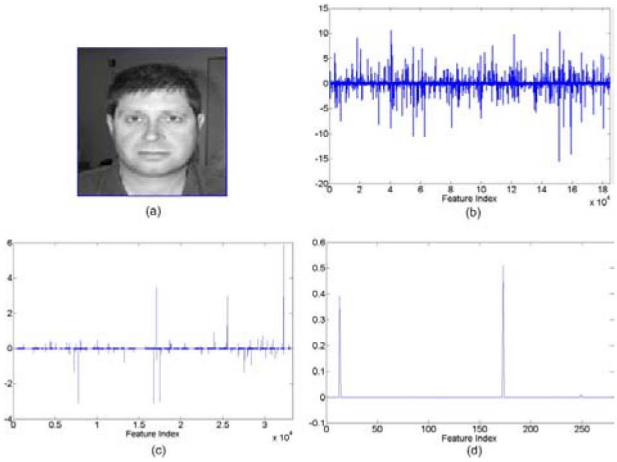


Fig. 13. Strength of coefficients $w_{nki}^{(l)}$ for all of three layers. (a) Image considered, (b) layer 1, (c) layer 2, (d) layer 3.

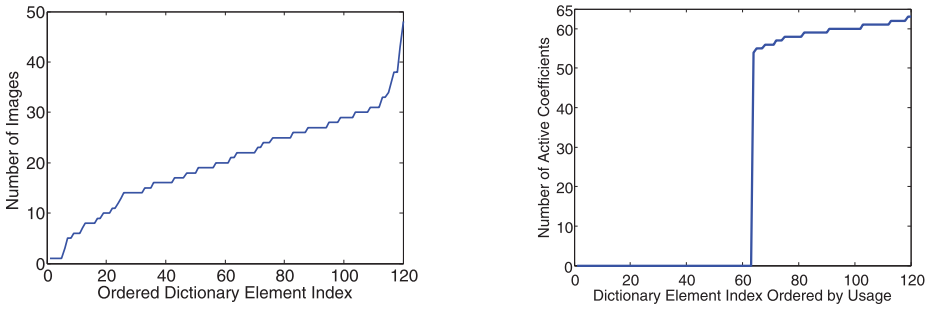
(a) (b) (c) (d)

Fig. 14. The reconstructed images at each layer for Fig. 9a. (a) Image after local contrast normalization, (b) reconstructed image at Layer 1, (c) reconstructed image at Layer 2, (d) reconstructed image at Layer 3.

to impose sparseness on hidden units [1], [7]. However, a detailed examination of the impact of sparseness on various terms of such models has received limited quantitative attention. In this section, we employ the Gibbs sampler and provide a detailed analysis on the effects of hyperparameters on model sparseness.

As indicated at the beginning of this section, parameter b (the IBP strength parameter) controls sparseness on the number of filters employed (via the probability of usage, defined by $\{\pi_k\}$). The normal-gamma prior on the w_{nki} constitutes a Student-t prior, and with $e = 1$, parameter f controls the degree of sparseness imposed on the filter usage (sparseness on the factor scores $\mathbf{W}_{nk}$); recall that within the prior $\alpha_{nki} \sim \text{Gamma}(e, f)$, with α_{nki} the precision for image n, dictionary element k, shift location i. Finally, the components of the filter $\boldsymbol{d}_k$ are also drawn from a Student-t prior, and with $g = 1$, h controls the sparsity of each $\boldsymbol{d}_k$. Below we focus on the impact of sparseness parameters b, d, and f on sparseness, and model performance, again showing Gibbs results (with very similar results manifested via batch and online VB).

In Fig. 15, we present variations of MSE with these hyperparameters, varying one at a time, and keeping the other fixed as discussed above. These computations were performed on the face Caltech 101 data using Gibbs computations, averaging 1,000 collection samples; 20 face images were considered and averaged over, and similar results were observed using other image classes. A wide range of these parameters yield similar good results, all favoring sparsity (note the axes are on a log scale). Note that as parameter b increases, a more-parsimonious (sparse) use of filters is encouraged, and as b increases



(a) (b)

Fig. 12. The statistics of Layer-2 coefficients (factor scores) on Caltech 101 face data. (a) The usage of Layer-2 dictionary elements across *all* held-out images; (b) the number of active coefficients of different Layer-2 dictionary elements for one (representative) held-out image. In this experiment, we randomly select 50 face images as training data to learn the dictionary and another different 50 images as held-out images to investigate the sparsity for different layers (shown in these figures). The number of Layer-2 dictionary elements is 120, and the size of each max-pooled coefficient matrix is 8×8.

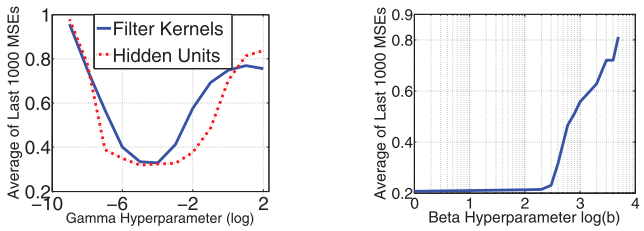


Fig. 15. Average MSE calculated from the last 1,000 Gibbs samples, considering BP analysis on the Caltech 101 faces data (averaged across 20 face images considered).

the number of inferred dictionary elements (at Layer 2 in Fig. 16) decreases.

5.6 Classification on Caltech 101

We consider the same classification problem considered by the authors of [1], [7], [31], [4], [32], considering Caltech 101 data [1], [7], [8]. The analysis is performed in the same manner these authors have considered previously, and therefore, our objective is to demonstrate that the proposed new model construction yields similar results. Results are based upon a Gibbs, batch VB, and online VB (minibatch size 20).

We consider a two-layer model, and for Gibbs, batch VB, and online VB, 1,020 images are used for training (such that the size of the training set is the same for all cases). We consider classifier design similar to that in [7], in which we perform classification based on layer-one coefficients, or based on both layers 1 and 2. We use a max-pooling step like that advocated in [11], [12], and we consider the image broken up into subregions as originally suggested in [31]. Therefore, with 21 regions in the image [31], and F features on a given level, the max-pooling step yields a $21 \cdot F$ feature vector for a given image and model layer ($42 \cdot F$ features when using coefficients from two layers). Using these feature vectors, we train an SVM as in [7], with results summarized in Table 2 (this table is from [1], now with inclusion of results corresponding to the method proposed here). It is observed that each of these methods yields comparable results, suggesting that the proposed method yields highly competitive performance; our classification results are most similar to the deep model considered in [7] (and, as indicated above, the descriptive form of our learned dictionary elements are also most similar to [7], which is based on an entirely different means of modeling each layer). Note, of course, that contrary to previous

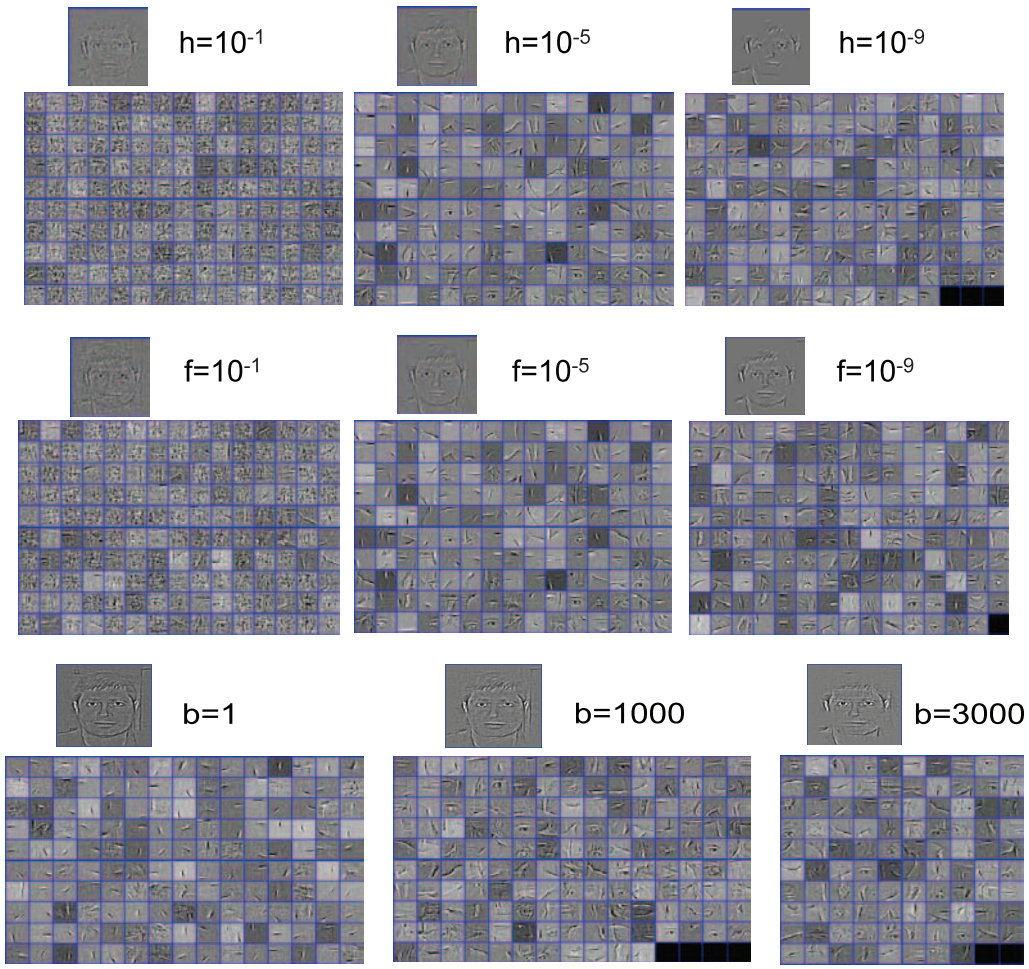


Fig. 16. Considering 20 face images from Caltech 101, we examine setting of sparseness parameters; unless otherwise stated, $b = 10^2$, $f = 10^{-5}$, and $h = 10^{-5}$. Parameters h and f are varied in (a) and (b), respectively. In (c), we set $e = 10^{-6}$ and $f = 10^{-6}$ and make hidden units unconstrained to test the influence of parameter b on the model's sparseness. In all of cases, we show the Layer-2 filters (ordered as above) and an example reconstruction.

TABLE 2
Classification Performance of the Proposed Model (Denoted cFA, for Convolutional Factor Analysis) with Several Methods from the Literature

# Training / Testing Examples	15/15	30/30
Bo *et al.* Layer-1+2 [34] (HMP)	–	$76.8 \pm 0.4\%$
Yu *et al.* Layer-1+2 [33] (HSC)	–	74.0%
Lazebnik *et al.* [31] (SPM)	56.4	$64.6 \pm 0.7\%$
Jarret *et al.* [4] (PSD)	–	$65.6 \pm 1.0\%$
Boureau *et al.* [36] (Macrofeatures)	–	$70.9 \pm 1.0\%$
Kavukcuoglu *et al.* Layer-1+2 [37](ConvSC)	–	$65.7 \pm 0.7\%$
Zeiler *et al.* Layer-1+2 [1] (DN)	$58.6 \pm 0.7\%$	$66.9 \pm 1.1\%$
Zeiler *et al.* Layer-1+2 [35] (AdaptiveDN)	–	$71.0 \pm 1.0\%$
Lee *et al.* Layer-1+2 [7] (CDBN)	$57.7 \pm 1.5\%$	$65.4 \pm 0.5\%$
cFA Layer-1 Gibbs sampler	$53.5 \pm 1.3\%$	$62.5 \pm 0.9\%$
cFA Layer-1+2 Gibbs sampler	$58.0 \pm 1.1\%$	$65.7 \pm 0.8\%$
cFA Layer-1 Batch VB	$52.1 \pm 1.5\%$	$61.2 \pm 1.2\%$
cFA Layer-1+2 Batch VB	$56.8 \pm 1.3\%$	$64.5 \pm 0.9\%$
cFA Layer-1 Online VB	$52.6 \pm 1.2\%$	$61.5 \pm 1.1\%$
cFA Layer-1+2 Online VB	$57.0 \pm 1.1\%$	$64.9 \pm 0.9\%$

Results are for the Caltech 101 dataset. The results are arranged in four groups, from top to bottom. Group 1: hierarchical sparse coding (HSC) based on multiscale raw patches with linear classifier; Group 2: other related methods with Spatial Pyramid Matching (SPM) classifier; Group 3: existing related convolutional sparse coding methods with SPM classifier; Group 4: our approach.

models, critical parameters such as dictionary size are automatically learned from the data in our model.

In Table 2, we show results on models of the type considered here, as well as the latest results from related models that have considered this dataset. There are four methods whose performances are beyond 70 percent, specifically Hierarchical Matching Pursuit (HMP), HSC, and Macrofeature, and Adaptive Deconvolutional Networks (AdaptiveDN). The first three methods are based on overlapping *local* patches (not explicitly convolutional). Macrofeature [11] embeds the sophisticated joint encoding scheme into the spatial pyramid framework via extracted SIFT type features with SPM classifier (rather than processing the raw pixels, as we do). In [33], for HSC, the authors proposed a jointly trained two-layer sparse coding framework, which accounts for high-order dependency among patterns in a local image neighborhood and captures spatial correlation via the variance of sparse coding responses. HMP [34] first learns a dictionary from a large set of image patches from the training images. After the dictionary is learned, in the first layer sparse codes are computed on small patches around each single pixel and then pooled into feature vectors representing patches by spatial pyramid max pooling. The second layer encodes these feature vectors via a dictionary learned from sampled patch level feature vectors. Thanks to good computational complexity, both HSC and HMP construct the very high-dimensional features at each layer with a large number of dictionary elements and finally use linear classifier.

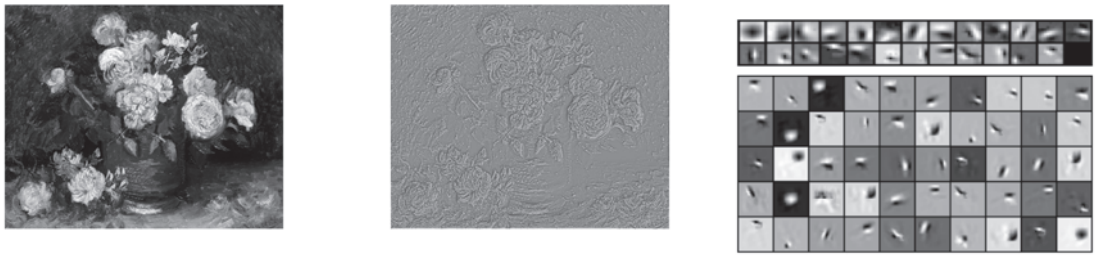
Most related to our model, AdaptiveDN [35], a multilayered convolutional network, overcomes the drawback of traditional layerwise stacked deep models that higher layers of the model have an increasingly diluted connection to the input. Instead, they locally adapt the models filters to the observed data and allow each layer to be trained with respect to the original image, rather than the output of the previous layer. But due to this adaptivity, the dictionary may need to change for different image classes, rather than being general. An approach of this form could be considered in the future for the proposed method.

For our methods, concerning performance comparisons between the different computational methods, the results based upon Gibbs sampling are consistently better than those based on VB, and the online VB results seem to generalize to held-out data slightly better than batch VB. The significant advantage of VB comes in the context of dictionary learning times. On 1,020 images for each layer, the Gibbs results (1,500 samples) required about 20 hours, while the batch VB on 1,020 images required about 11 hours. By contrast, when analyzing all 9,144 Caltech 101 images, online VB required less than 3 hours. All computations were performed in Matlab, on an Intel Core i7 920 2.26 GHz and 6-GB RAM computer (as considered in all experiments).

5.7 Online VB and Van Gogh Painting Analysis

Our final experiment is on a set of high-resolution scans of paintings by Vincent van Gogh. This dataset includes 101 high resolution paintings with various sizes (e.g., $5{,}000 \times 5{,}000$). Art historians analyze the paintings locally, dividing into subimages; we follow the same procedure and divide each image into several 128×128 smaller images (each painting ranged from 16 to 2,000 small images) [38]. The total dataset size is 40,000 subimages.

One issue with these paintings is that most of the time a significant portion of the image does not include any



(a) Original Image (b) Pre-processed Image (c) Online Dictionary

Fig. 17. Analyzing Van Gogh paintings with Online VB. (a) The original image, (b) preprocessing highlights the brushstrokes (best viewed by electronic zooming), (c) first (top) and second (bottom) layer dictionaries trained via Online VB by processing 20,000 subimages.

brushstroke; it is mostly insignificant background including cracks of paint due to aging or artifacts from canvas. This can be observed in Fig. 17. Batch deep learning cannot handle this dataset due to its huge size. Online algorithm provides a way to train the dictionary by going over the entire dataset: iteratively sampling several portions from paintings and analyzing the entire dataset. This mechanism also provides robustness to the regions of paintings with artifacts—even if some subimages are from such a region, the ability to go over the entire dataset eliminates overfitting those.

For the first and second layers, we use dictionary element sizes of 9×9 and 3×3, and corresponding max-pooling ratios of 5 and 2. The minibatch size is 6. The truncation level K is set to 25 for the first layer and 50 for the second layer. The learning rate parameters are $\rho_0 = 1$ and $\kappa = 0.5$. We analyzed 20,000 subimages and the learned dictionary is shown in Fig. 17. The layer-2 outputs, when viewed on the image plane, look similar to the basic vision tokens discussed by Marr [39] and Donoho [40]. This is consistent with our observation in Fig. 10a and one of the key findings of [20]: When the diversity in the dataset is high (e.g., analyzing all classes of Caltech 101 together), learned dictionaries look like primitive edges. We observe that similar type of tokens are extracted by analyzing diverse brushstrokes of Van Gogh which have different thicknesses and directions.

6 Conclusions

The development of deep unsupervised models has been cast in the form of hierarchical factor analysis, where the factor loadings are sparse and characterized by a unique convolutional form. The factor scores from layer l serve as the inputs to layer $l+1$, with a max-pooling step. By framing the problem in this manner, we can leverage significant previous research with factor analysis [41]. Specifically, a truncated beta-Bernoulli process [16], [17], motivated by the IBP [15], has been used on the number of dictionary elements needed at each layer in the hierarchy (with a particular focus on inferring the number of dictionary elements at higher levels, while at layer 1 we typically set the model with a relatively small number of primitive dictionary elements). We also employ a hierarchical construction of the Student-t distribution [23] to impose sparseness on the factor scores, with this related to previous sparseness constraints imposed via ℓ_1 regularization [1]. The inferred dictionary elements, particularly at higher levels, typically have descriptive characteristics when viewed in the image plane.

Summarizing observations, when the model was employed on a specific class of images (e.g., cars, planes, seats, motorcycles, etc.), the Layer-3 dictionary elements when viewed in the image plane looked very similar to the associated class of images (see Fig. 8), with similar (but less fully formed) structure seen for Layer-2 dictionary elements. Further, for such single-class studies the beta-Bernoulli construction typically only used a relatively small subset of the total possible number of dictionary elements at Layers 2 and 3, as defined by the truncation level. However, when considering a wide class of "natural" images at once, while the higher level dictionary elements were descriptive of localized structure within imagery, they looked less like any specific entity (see Fig. 12); this is to be expected by the increased heterogenity of the imagery under analysis. Additionally, for the dictionary-element truncation considered, as the diversity in the images increased it was more likely that all of the possible dictionary elements within the truncation level were used by at least one image. Nevertheless, as expected from the IBP, there was a relatively small subset of dictionary elements at each level that were "popular", or *widely* used (see Section 3.1).

In general, we find that the posterior distribution on the number of dictionary elements may change as a function of the type of images considered. However, based upon running the IBP framework on a given class of images, one can safely set the number of dictionary elements at the mode of the posterior. For similar images in the future one may use that setting and avoid the IBP. We would argue that the IBP provides a "principled means of model exploration. One could alternatively explore (with perhaps exhaustive testing) different settings for the number of dictionary elements, but the proposed approach is more systematic, and perhaps valuable when new classes of images are considered.

References

[1] M. Zeiler, D. Krishnan, G. Taylor, and R. Fergus, "Deconvolution Networks," *Proc. IEEE Conf. Computer Vision Pattern Recognition*, 2010.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[3] G. Hinton and R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, pp. 504-507, 2006.

[4] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What Is the Best Multi-Stage Architecture for Object Recognition?" *Proc. IEEE Int'l Conf. Computer Vision,* 2009.

[5] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun, "Efficient Learning of Sparse Representations with an Energy-Based Model," *Proc. Neural Information Processing Systems,* 2006.

[6] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol, "Extracting and Composing Robust Features with Denoising Autoencoders," *Proc. Int'l Conf. Machine Learning,* 2008.

[7] H. Lee, R. Grosse, R. Ranganath, and A.Y. Ng, "Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations," *Proc. Int'l Conf. Machine Learning,* 2009.

[8] H. Lee, Y. Largman, P. Pham, and A. Ng, "Unsupervised Feature Learning for Audio Classification Using Convolutional Deep Belief Networks," *Proc. Neural Information Processing Systems,* 2009.

[9] M.R.M. Norouzi and G. Mori, "Stacks of Convolutional Restricted Boltzmann Machines for Shift-Invariant Feature Learning," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2009.

[10] H. Lee, C. Ekanadham, and A.Y. Ng, "Sparse Deep Belief Network Model for Visual Area V2," *Proc. Neural Information Processing Systems,* 2008.

[11] Y. Boureau, F. Bach, Y. LeCun, and J. Ponce, "Learning Mid-Level Features for Recognition," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2010.

[12] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong, "Locality-Constrained Linear Coding for Image Classification," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2010.

[13] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A Theoretical Analysis of Feature Pooling in Vision Algorithms," *Proc. Int'l Conf. Machine Learning,* 2010.

[14] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online Dictionary Learning for Sparse Coding," *Proc. Int'l Conf. Machine Learning,* 2009.

[15] T.L. Griffiths and Z. Ghahramani, "Infinite Latent Feature Models and the Indian Buffet Process," *Proc. Advances in Neural Information Processing Systems,* pp. 475-482, 2005.

[16] R. Thibaux and M. Jordan, "Hierarchical Beta Processes and the Indian Buffet Process," *Proc. Int'l Conf. Artificial Intelligence and Statistics,* 2007.

[17] J. Paisley and L. Carin, "Nonparametric Factor Analysis with Beta Process Priors," *Proc. Int'l Conf. Machine Learning,* 2009.

[18] M. Zhou, H. Chen, J. Paisley, L. Ren, G. Sapiro, and L. Carin, "Non-Parametric Bayesian Dictionary Learning for Sparse Image Representations," *Proc. Neural Information Processing Systems,* 2009.

[19] R. Adams, H. Wallach, and Z. Ghahramani, "Learning the Structure of Deep Sparse Graphical Models," *Proc. Int'l Conf. Artificial Intelligence and Statistics,* 2010.

[20] B. Chen, G. Polatkan, G. Sapiro, D. Dunson, and L. Carin, "The Hierarchical Beta Process for Convolutional Factor Analysis and Deep Learning," *Proc. Int'l Conf. Machine Learning,* 2011.

[21] M. West, "Bayesian Factor Regression Models in the 'Large p, Small n' Paradigm," *Bayesian Statistics 7,* J.M. Bernardo, M. Bayarri, J. Berger, A. Dawid, D. Heckerman, A. Smith, and M. West, eds., pp. 723-732, Oxford Univ. Press, 2003.

[22] C. Carvalho, J. Chang, J. Lucas, J.R. Nevins, Q. Wang, and M. West, "High-Dimensional Sparse Factor Modelling: Applications in Gene Expression Genomics," *J. Am. Statistical Assoc.,* vol. 103, pp. 1438-1456, 2008.

[23] M. Tipping, "Sparse Bayesian Learning and the Relevance Vector Machine," *J. Machine Learning Research,* vol. 1, pp. 211-244, 2001.

[24] R. Adams, H. Wallach, and Z. Ghahramani, "Learning the Structure of Deep Sparse Graphical Models," *Proc. Int'l Conf. Artificial Intelligence and Statistics,* 2010.

[25] M. Hoffman, D. Blei, and F. Bach, "Online Learning for Latent Dirichlet Allocation," *Proc. Advances in Neural Information Processing Systems,* 2010.

[26] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines.* Cambridge Univ. Press, 2000.

[27] R.C.J. Weston and F. Ratle, "Deep Learning via Semi-Supervised Embedding," *Proc. Int'l Conf. Machine Learning,* 2008.

[28] Y.-L. Boureau, M. Ranzato, and F.-J. Huang, "Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2007.

[29] P.O. Hoyer, "Non-Negative Matrix Factorization with Sparseness Constraints," *J. Machine Learning Research,* vol. 5, pp. 1457-1469, 2004.

[30] D.D. Lee and H.S. Seung, "Learning the Parts of Objects by Non-Negative Matrix Factorization," *Nature,* vol. 401, no. 6755 pp. 788-791, 1999.

[31] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2006.

[32] H. Zhang, A.C. Berg, M. Maire, and J. Malik, "SVM-KNN: Discriminative Nearest Neighbor Classification for Visual Category Recognition," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2006.

[33] K. Yu, Y. Lin, and J. Lafferty, "Learning Image Representations from the Pixel Level via Hierarchical Sparse Coding," *Proc. IEEE Conf, Computer Vision and Pattern Recognition,* Dec. 2011.

[34] L. Bo, X. Ren, and D. Fox, "Hierarchical Matching Pursuit for Image Classification: Architecture and Fast Algorithms," *Proc. Advances in Neural Information Processing Systems,* Dec. 2011.

[35] M.D. Zeiler, G.W. Taylor, and R. Fergus, "Adaptive Deconvolutional Networks for Mid and High Level Feature Learning," *Proc. IEEE Int'l Conf. Computer Vision,* 2011.

[36] Y. Boureau, F. Bach, Y. LeCun, and J. Ponce, "Learning Mid-Level Features for Recognition," *Proc. IEEE Conf. Computer Vision and Pattern Recognition,* 2010.

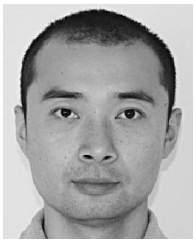
[37] K. Kavukcuoglu, P. Sermanet, Y. Boureau, K. Gregor, M. Mathieu, and Y. LeCun, "Learning Convolutional Feature Hierachies for Visual Recognition," *Proc. Advances in Neural Information Processing Systems,* 2010.

[38] C.R. Johnson, E. Hendriks, I. Berezhnoy, E. Brevdo, S.M. Hughes, I. Daubechies, J. Li, E. Postma, and J.Z. Wang, "Image Processing for Artist Identification: Computerized Analysis of Vincent van Gogh's Brushstrokes," *IEEE Signal Processing Magazine,* vol. 25, no. 4, pp. 37-48, July 2008.

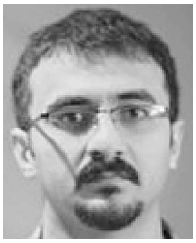
[39] D. Marr, *Vision.* Freeman, 1982.

[40] D.L. Donoho, "Nature vs. Math: Interpreting Independent Component Analysis in Light of Recent work in Harmonic Analysis," *Proc. Int'l Workshop Independent Component Analysis and Blind Signal Separation,* pp. 459-470, 2000.

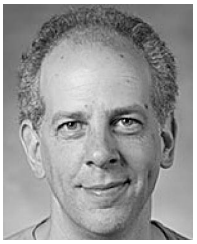
[41] D. Knowles and Z. Ghahramani, "Infinite Sparse Factor Analysis and Infinite Independent Components Analysis," *Proc. Seventh Int'l Conf. Independent Component Analysis and Signal Separation,* 2007.



Bo Chen received the PhD degree in electrical engineering from Xidian University in China. He is a postdoctoral researcher in the Electrical and Computer Engineering Department at Duke University. He is a member of the IEEE.

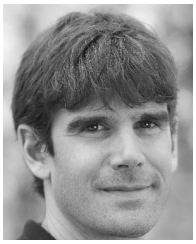


Gungor Polatkin is a graduate student in the Electrical and Computer Engineering Department at Princeton University.



Guillermo Sapiro received the PhD degree in electrical engineering from the Technion in Israel. He is a professor in the Electrical and Computer Engineering Department at Duke University. He is a senior member of the IEEE.

David Blei received the PhD degree in electrical engineering from the University of California, Berkeley. He is an associate professor in the Computer Science Department at Princeton University. He is a member of the IEEE.



David Dunson received the PhD degree in statistics from Emory University. He is a professor in the Statistics Department at Duke University.

Lawrence Carin received the PhD degree in electrical engineering from the University of Maryland. He is a professor in the Electrical and Computer Engineering Department at Duke University. He is a fellow of the IEEE and a member of the IEEE Computer Society.