Learning The Structure of Deep Convolutional Networks

Jiashi Feng
Department of EECS & ICSI
UC Berkeley
jshfeng@berkeley.edu

Trevor Darrell
Department of EECS & ICSI
UC Berkeley
trevor@eecs.berkeley.edu

Abstract

In this work, we develop a novel method for automatically learning aspects of the structure of a deep model in order to improve its performance, especially when labeled training data are scarce. We propose a new convolutional neural network model with the Indian Buffet Process (IBP) prior, termed ibpCNN. The ibpCNN automatically adapts its structure to provided training data, achieves an optimal balance among model complexity, data fidelity and training loss, and thus offers better generalization performance.

The proposed ibpCNN captures complex data distribution in an unsupervised generative way. Therefore, ibpCNN can exploit unlabeled data – which can be collected at low cost – to learn its structure. After determining the structure, ibpCNN further learns its parameters according to specified tasks, in an end-to-end fashion, and produces discriminative yet compact representations.

We evaluate the performance of ibpCNN, on fully- and semi-supervised image classification tasks; ibpCNN surpasses standard CNN models on benchmark datasets with much smaller size and higher efficiency.

1. Introduction

Deep learning models, and in particular convolutional neural networks (CNNs) [17], have achieved very good performance in a number of benchmarks [15, 8, 10]. The performance gain brought by deep models arguably lies in their end-to-end learning strategy, multi-layer architecture and the availability of sufficiently large training datasets.

Several recent works [22, 24] suggest when they grow larger, the networks will achieve better performance. However, a very large neural network, which thus has great complexity, generally demands significantly more training data for learning its free parameters in order to obtain satisfactory generalization performance. Therefore, in the absence of sufficient training data, a complicated network model may not perform comparably well as a simpler model for some tasks [18]. Hence, choosing an appropriate and compact structure for a network, which involves making an optimal trade-off between the amount of available training data and model complexity, is desirable for obtaining good practical performance.

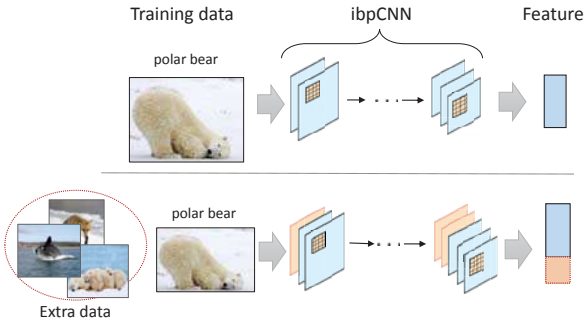

Figure 1. The number of filters in each layer of the proposed ibpCNN model is adaptive to the complexity of training data: an ibpCNN trained on a relatively small dataset (top panel) has a smaller size than the one trained on a lager dataset including some extra data (bottom panel). Thus the output representations grow in complexity with input data.

In this work, we propose a novel Convolutional Neural Network model equipped with an Indian Buffet Process prior [9], called ibpCNN, to solve this important problem. As shown in Figure 1, the ibpCNN systematically evolves from a simplest network, driven by the provided training data and any specified task, and finally converges to a compact model of an appropriate structure. During the evolution the ibpCNN only requires training data along with loss functional characterizing the targeted task as inputs, and does not require explicitly trying multiple different structures via cross-validation.

Automatically learning the structure of a deep model is a difficult task, due to the huge search space of possible models. Fortunately, model structure learning has been elegantly explored by several generative model priors, such as the Indian Buffet Process (IBP) [9, 1] or Beta Process (BP) [21]. These priors provide a systematic and principled approach to search the model space. In this work, we also

follow this paradigm and perform structure learning using a nonparametric deep model, which exploits the layer-wise IBP prior.

More concretely, the proposed ibpCNN model introduces nonparametric convolutional and fully connected layers, whose dimensions are fully adaptive to the provided data and specified tasks. In its training phase, the layer size and filter parameters of ibpCNN are jointly tuned to reliably model the distribution of input data as well as minimize task-specific loss functionals. Such an end-to-end learning strategy enables ibpCNN to achieve optimal balance between data amounts and model complexity, and thus improve upon generalization performance of standard CNN models, especially when the training data are scarce.

In this work, we propose a novel Grow-And-Prune (GAP) algorithm to optimize the structure of each IBP layer in ibpCNN, during the training process. The GAP algorithm conducts complementary model growing and pruning to find optimal layer configuration efficiently, and is guaranteed to converge to the optimal layer configuration.

We demonstrate the applicability and effectiveness of ibpCNN for image classification tasks. Comprehensive experiments on benchmark datasets show that the end-to-end trained ibpCNN is indeed much more compact than standard CNN models, and more efficient for prediction.

In short, the proposed ibpCNN has the following benefits: (1) its size automatically adapts to the complexity of the training data and specific tasks; (2) it offers more compact image representations and higher efficiency in prediction; (3) it is able to exploit unlabeled data to assist modeling complicated data distribution; and (4) it achieves better generalization performance when training data are limited.

2. Related Works

Although very successful when provided very large labeled datasets [15, 8], CNN models may generalize poorly on smaller datasets because they require the estimation of millions of parameters. While there has already been some work on using deep learning methods for attribute prediction [5], we explore alternative ways to automatically determine the appropriate number of filters in the deep neural networks.

The proposed ibpCNN models conduct input decomposition in convolutional and fully connected manners, which is similar to the deconvolutional networks [27, 28], Predictive Sparse Decomposition (PSD) [14, 12] and convolutional auto-encoders [19]. However, there are two important differences between ibpCNN and those models. First, ibpCNN adapts its structure to the data complexity and secondly ibpCNN can be trained in an end-to-end manner.

Several works focused on removing redundant filters from a well trained network of large size. For example, Jaderberg *et al.* [11] proposed to reduce the redundancy of filters by exploiting their low rank combination, and Le Cun *et al.* [18] proposed to find the redudant the filters by examining their gradient magnitudes. Zhou *et al.* [29] also developed a greedy method to determine the suitable number of filters for denoising auto-encoders. Different from those methods, our proposed ibpCNN model adapts the network structure in its growing process, instead of via ad-hoc post processing, with performance guarantee based on submodular optimization technique.

3. Preliminaries: Indian Buffet Process

In this section, we briefly review the Indian Buffet Process (IBP) model [9] and its small variance asymptotics. The asymptotic IBP allows the structure of a model to be adaptive to data complexity and produces compact feature representations. It serves as the foundation to construct our proposed ibpCNN model. More details on the derivations of asymptotic IBP models are provided in the supplementary material due to space limit.

IBP is a nonparametric prior for describing an *infinite* latent feature model, which assumes there are infinitely many feature basis for representing input samples, and the basis distribution follows the IBP prior. More concretely, let $x \in \mathbb{R}^d$ denote an input sample, $W \in \mathbb{R}^{d\times K}$ incorporates the basis for representing x, and denote the resulted representations of x w.r.t. the basis W as $z \in \mathbb{R}^K$, where the value of K can be infinite [9]. The original IBP model is rather computationally expensive for optimization. As an alternative, Broderick *et al*. [3] derive the following small variance approximation of the joint distribution $P(x, W, z)$:

$$-\log p(x, W, z) \sim \sum_{i=1}^{N} \|x_i - W z_i\|_2^2 + \lambda^2 K, \quad (1)$$

up to constant terms. This is the probability function we are going to work with in the proposed ibpCNN.

4. IBP Convolutional Neural Networks

We develop a convolutional neural network based on the above asymptotic IBP model – the ibpCNN. We first present unsupervised ibpCNN layers, similar in goal as convolutional auto-encoders [19] but with analytic derivations. We then in Section 4.2 and Section 4.2 present fully- and semi-supervised training of ibpCNN. Details about parameter optimization of ibpCNN follow in the subsequent section.

In particular, we use $X^{(\ell)}$ and $Z^{(\ell)}$ to denote the tensor of input and output feature maps of the ℓ-th layer in ibpCNN respectively. The filters in the ℓ-th layer are parameterized by a tensor $W^{(\ell)}$. The number of filters in the ℓ-th layer is denoted as $K^{(\ell)}$, which is going to be inferred in the training phase. We use $z_k^{(\ell)}$ to denote an output feature map in the k-th channel and $w_k^{(\ell)}$ denotes the corresponding filter

producing the feature map. When clear from context, we omit the superscript (ℓ) to avoid heavy notations.

4.1. Unsupervised IBP Layers

In a *fully connected* layer of ibpCNN, the i-th input feature vector x_i of the i-th sample, which is vectorized from its feature map X_i, is reconstructed by a linear combination of the filters $W = [w_1, \ldots, w_K]$ and output feature map z_i of dimension K following the asymptotic model in Eqn. (1). The unsupervised loss in fully connected IBP layers is then defined as:

$$\mathcal{L}_{\text{fc}}(W, z, K) := \sum_i \|x_i - W z_i\|^2 + \lambda^2 K. \qquad (2)$$

Here the number of filters K is optimized to achieve the best balance between the representative ability (minimizing the reconstruction loss in the first term) and complexity of the layer (the second term). We will elaborate more on how to set the trade-off parameter λ in Section 6.

The above fully connected IBP layer differs from its counterpart in a CNN model: the IBP fully connected layer decomposes the input x over a learned basis W in order to minimize the reconstruction error (captured by the first term in the loss) and the model complexity (the second term); it thus pursues a compact representation of the input by learning a set of appropriate basis[1].

We now proceed to define the *convolutional* layers in ibpCNN. Similar to fully connected layers, the loss function of convolutional layer, $\mathcal{L}_{\text{conv}}$, also comprises two terms: a likelihood term that keeps the reconstruction close to the original input, and a regularization term that penalizes the model complexity to make sure the representation to be compact. The filter parameters W and output response maps Z, along with the number of filters K, are inferred by minimizing following loss:

$$\mathcal{L}_{\text{conv}}(W, Z, K) := \sum_i \|X_i - \sum_{k=1}^{K} w_k * Z_{i,k}\|^2 + \lambda^2 K. \quad (3)$$

Here $Z_{i,k}$ – the feature map of the i-th sample at the k-th channel – takes the role of a feature map which is convolved with a filter w_k and added over all k to approximate the input signal X, and $*$ denotes a convolution operation.

The convolutional IBP layer is similar to the deconvolutional network proposed in [27] where the cost to minimize at each layer is also the mean square error on the inputs. However, unlike the deconvolutional network, our proposed IBP convolutional layer does not use any form of sparse coding and does not have fixed size. Applying this data-driven layer construction to natural images produces a compact set of filters that capture data specific high-order image structure beyond edge primitives.

[1] If we further impose the orthogonal constraint on W: $W^\top W = I$, the IBP layer will degenerate to Independent Component Analysis, which can be optimized more efficiently at the cost of losing representative power.

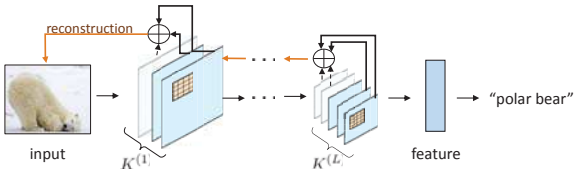

Figure 2. Illustration on the architecture of ibpCNN with L layers. The number of filters per layer (*i.e.*, $K^{(1)}, \ldots, K^{(L)}$) are adaptive to the provided training data. When a dataset becomes more complicated, some new filters (in dashed line) will be automatically introduced into ibpCNN. In a specific layer ℓ, the inputs are reconstructed (orange line) via the linear combination of $K^{(\ell)}$ feature maps convolved with learned filters. The outputs of the final layer pass through a fully connected layer to produce classification results.

4.2. Supervised Training for ibpCNN

We now describe the overall architecture of ibpCNN, and explain how to train ibpCNN for supervised learning tasks. The ibpCNN model is constructed by stacking multiple convolutional and fully connected IBP layers, and its structure is determined by both the provided data and the specified tasks. Thus ibpCNN is different from canonical deep generative models, such as convolutional auto-encoder [19]. A graphical illustration for the structure of ibpCNN is provided in Figure 2.

Suppose the ibpCNN consists of L_{conv} convolutional layers and L_{fc} fully connected layers and denote $L = L_{\text{conv}} + L_{\text{fc}}$, then the loss function of multi-layered ibpCNN to minimize is:

$$\sum_{\ell=1}^{L} \mathcal{L}_{\text{task}}(Z^{(\ell)}; X, y) + \sum_{\ell=1}^{L_{\text{conv}}} \mathcal{L}_{\text{conv}}(W^{(\ell)}, Z^{(\ell)}, K^{(\ell)}; X^{(\ell-1)}) + \sum_{\ell=L_{\text{conv}}+1}^{L} \mathcal{L}_{\text{fc}}(W^{(\ell)}, Z^{(\ell)}, K^{(\ell)}; X^{(\ell-1)}), \qquad (4)$$

where $X^{(0)} = X$ denotes the raw input images and the input of the $(\ell-1)$-st layer $X^{(\ell-1)}$ is derived from applying proper normalization and pooling operations on the output $Z^{(\ell-1)}$ of the ℓ-th layer. Here y is the ground truth annotation for the provided supervised training examples and $\mathcal{L}_{\text{task}}$ is a loss function of a multi-layered network designed for the targeted task. For example, if targeting at image classification, $\mathcal{L}_{\text{task}}$ can be the combination of softmax and cross-entropy [15]: $\mathcal{L}_{\text{task}} = \sum_{i=1}^{n} \left\langle y_i, \log \frac{\exp(Z^{(L)})}{\|\exp(Z^{(L)})\|_1} \right\rangle$, with n supervised training examples and $Z^{(L)}$ depending on $W^{(\ell)}, Z^{(\ell)}|_{\ell=1}^{L-1}$ and X. Thus, the number of filters K in all the IBP layers is tuned to minimizing both the training data and the task related loss, through the end-to-end optimization. That is why ibpCNN is constructed in a joint of data

and task driven manner. We can also generalize the loss function $\mathcal{L}$ beyond classification loss, and adapt the structure of ibpCNN to other tasks (*e.g.*, object segmentation).

4.3. Semi-supervised Training for ibpCNN

The lack of well annotated training data is a common challenge in many machine learning problems, especially for training deep neural networks with a huge number of parameters. In contrast to the expensive cost for labeling training data, unlabeled data from the same domain usually can be collected at almost zero cost. Therefore, semi-supervised learning becomes a promising solution to the problems caused by insufficient supervision information.

The proposed ibpCNN can be naturally applied for semi-supervised learning. In semi-supervised learning, we are provided with data that appear as labeled pairs $\{(X_1, y_1), \ldots, (X_n, y_n)\}$ and unlabeled data $\{X_{n+1}, \ldots, X_{n+n_1}\}$. The proposed ibpCNN effectively employs semi-supervised learning to exploit generative descriptions of the combined data $\{X_1, \ldots, X_n, X_{n+1}, \ldots, X_{n+n_1}\}$ in its structure learning phase (step (a) in Algorithm 2). Thus ibpCNN is capable of improving upon the classification performance that would be obtained using the labeled data alone, and offering better generalization performance, especially when the number of labeled training data is rather limited ($n \ll n_1$). We verify this advantage of ibpCNN in the experiments presented in Section 6. This semi-supervised learning strategy is similar to the layer-wise greedy pre-training in generative model such as deep belief networks (DBNs) [2], which also adapts the network parameters to provided data without supervision information. In addition to adjusting parameters, ibpCNN also adapts its structure to both the labeled and unlabeled data.

5. Optimization

This section presents details regarding training procedure of the ibpCNN. The three loss terms of ibpCNN in Eqn. (4) are minimized alternatively: we first fix the filter parameters $W^{(\ell)}$ and optimize the structure of the ibpCNN (i.e., the parameters $K^{(\ell)}$) through applying a novel grow-and-prune algorithm to minimize the latter two data reconstruction loss terms; then we fix the structure of the network, and update the parameters $W^{(\ell)}$ through minimizing the task loss term via back propagation; finally the layer-wise outputs $Z^{(\ell)}$ are inferred to minimize the reconstruction loss again, according to the updated parameters and network structure. Following subsections explain the details about the structure learning, data representation inference and filter parameter optimization respectively.

5.1. Learning The Structure of ibpCNN

The most straightforward method to determine the structure of a layer is forward greedy selection: one can start with the simplest structure consisting of a single filter (*i.e.*, $K = 1$) and greedily increase the number of filters until the loss function in Eqn. (2) or (3) does not decrease anymore [3]. However, we empirically found such a simple greedy strategy is prone to being stuck at a local optimum in practice, due to its sensitiveness to the initial filter and the order of adding filters into the layer. The resulted layer is usually either over-complete or over-succinct and has deteriorated performance.

To avoid this issue, we propose a Grow-And-Prune (GAP) algorithm, which consists of two complementary operations: *grow* for increasing the complexity of the layer to produce more information-preserving representations of the inputs and *prune* to remove redundant filters in the layer and keep the layer compact. With a specified large number of filters (say N), minimizing the loss in Eqn. (2) or (3) provides an over-complete filter set. Then GAP starts with this over-complete and an empty filter set, performs layer growing and pruning from dual directions and finally converges to an optimal configuration of layer size. Intuitively, the proposed GAP is a "double" greedy algorithm. Different from a simple forward (resp. backward) greedy algorithm that starts from an empty (resp. full) set and iteratively adds (resp. removes) one filter to maximize the gain in optimization, the double greedy [4] GAP algorithm selects the appropriate and non-redundant filters from two complementary directions and thus it can effectively avoid being stuck at a local optimum, as explained in Section 5.4. More details of GAP are provided in Algorithm 1. At the beginning of ibpCNN training, the candidate filters $\mathcal{W}$ are randomly initialized.

GAP examines all the N filters for only one round and evaluates each filter by comparing its contribution in the grow step with the prune step. Such evaluations and comparisons do not require to re-optimize the loss function for every configuration of the layer size, and hence very efficient. Therefore, GAP is able to find the appropriate structure of one layer rather efficiently with a low computational complexity of $O(N^2m+Nmd)$. Here m and d are the number and dimension of inputs respectively. Applying GAP repeatedly through L layers constructs the whole ibpCNN.

The prune step in GAP shares similar spirit with the Optimal Brain Damage (OBD) strategy proposed in [18], which reduces the network size from an over-complete one by investigating the redundancy among filters. Different from OBD, GAP considers both layer growing and pruning in a complementary manner. The convergence guarantee of GAP is provided in a following subsection.

5.2. Inference

For a specific layer ℓ, inference involves finding the feature maps $Z^{(\ell)}$ that minimize $\mathcal{L}_{\text{fc}}$ (in Eqn. (2)) or $\mathcal{L}_{\text{conv}}$ (in Eqn. (3)), given an input feature map $X^{(\ell)}$ and filters $W^{(\ell)}$. $Z^{(\ell)}$ is inferred only from the input signal $X^{(\ell-1)}$ and basis $W^{(\ell)}$ of the same layer. However, since $W^{(\ell)}$ is tuned to task-related top-down information, $Z^{(\ell)}$ also implicitly depends on the top-down information. The feature maps $Z^{(\ell)}$ of different samples can be inferred in parallel. We adopt Accelerated Proximal Gradient (APG) descent [20] to infer $Z^{(\ell)}$. APG, as a first order method, does not involve matrix inverse and offers a fast convergence rate of $O(1/t^2)$ with t being the number of iterations, and it is efficient in inferring the output feature maps.

5.3. Parameter Learning

The goal of parameter learning is to fine tune the filters $W^{(\ell)}$ in ibpCNN, which are shared across all the provided training data. We use the standard back propagation (backprop) algorithm [17] to optimize weight parameters of all the filters in ibpCNN. During the parameter learning, both the task specific loss at the top layer in ibpCNN and the reconstruction loss in each specific layer are back-propagated to update the filter parameters. Thus, although they are initialized in an unsupervised learning manner (see Section 5.1), the parameters of ibpCNN are eventually fine tuned to the specific task to generate more discriminative representations.

Algorithm 2 gives the details on training the overall ibpCNN, which alternatively learns the structure of ibpCNN, infers the output feature maps and learns filter parameters. In the implementation, we allow up to $T' = 10$ and $T = 3{,}000$ epochs of the steps (b) and (c) before performing the layer structure learning in step (a), to ensure the ibpCNN to converge properly.

5.4. Notes on Convergence

The GAP algorithm is inspired by the recently developed "double greedy" algorithm [4] in the submodular optimization literatures. An appealing advantage of the GAP algorithm over simple greedily growing algorithms is its ability of converging to the optimal configurations, up to a constant factor. This is stated in the following theorem.

Theorem 1. *The output $\mathcal{U}_N$ of Algorithm 1 achieves the loss approximating the optimum with a constant factor of $1 < c \leq 2$:*

$$\mathcal{L}(\mathcal{U}) \leq c\mathcal{L}(\text{OPT}),$$

where OPT *represents the optimal configuration.*

The preliminaries about double greedy algorithms and proof on the above theorem are deferred to the supplementary material due to space limit.

Algorithm 1: Grow-And-Prune (GAP) for layer structure learning

Input : Maximal layer complexity N, loss function $\mathcal{L}$ ($\mathcal{L}_{\text{fc}}$ or $\mathcal{L}_{\text{conv}}$), a candidate set of filters $\mathcal{W} = \{w_1, \ldots, w_N\}$.
Initialization: An empty layer $\mathcal{U}_0 = \varnothing$, and a virtual over-complete layer $\mathcal{V}_0 = \mathcal{W}$.
for $i = 1$ *to* N **do**
  $p_i \leftarrow \mathcal{L}(\mathcal{U}_{i-1}) - \mathcal{L}(\mathcal{U}_{i-1} \cup \{w_i\})$.
  $q_i \leftarrow \mathcal{L}(\mathcal{V}_{i-1}) - \mathcal{L}(\mathcal{V}_{i-1} \backslash \{w_i\})$.
  $p_i' \leftarrow \max\{p_i, 0\}, q_i' \leftarrow \max\{q_i, 0\}$.
  with probability $p_i'/(p_i' + q_i')$
  `// Grow layer`
  $\mathcal{U}_i \leftarrow \mathcal{U}_{i-1} \cup \{w_i\}, \mathcal{V}_i \leftarrow \mathcal{V}_{i-1}$
  else
  `// Prune layer`
  $\mathcal{U}_i \leftarrow \mathcal{U}_{i-1}, \mathcal{V}_i \leftarrow \mathcal{V}_{i-1} \backslash \{w_i\}$.
end
Output: An IBP layer consisting of filters in $\mathcal{U}_N$.

Algorithm 2: Training for ibpCNN

input : Labeled training data $\{(X_1, y_1), \ldots, (X_n, y_n)\}$ and unlabeled training data (if any) $\{X_{n+1}, \ldots, X_{n+n_1}\}$, number of layers L, epochs T and T', parameters $\lambda^{(\ell)}$.
for t' *from* 1 *to* T' **do**
  Step (a): Apply GAP (Algorithm 1) on $\{X_1, \ldots, X_n\} \cup \{X_{n+1}, \ldots, X_{n+n_1}\}$ to learn the structures of ibpCNN.
  for t *from* 1 *to* T **do**
    Step (b): Fix $K^{(\ell)}$ and $Z^{(\ell)}$, apply backprop on $\{(X_1, y_1), \ldots, (X_n, y_n)\}$ to update filters $W^{(\ell)}, \forall \ell = 1, \ldots, L$.
    Step (c): Fix $W^{(\ell)}$ and $K^{(\ell)}$, perform inference to update output feature map $Z^{(\ell)}$, $\forall \ell = 1, \ldots, L$.
  end
end
output: The trained ibpCNN

6. Experiments

6.1. Datasets and Experiment Settings

Datasets We evaluate the performance of ibpCNN on following two benchmark datasets. The first is the Animal with Attributes (AwA) dataset, which is a collection of 9,380 images from 50 various animal categories. Each image is associated with annotations for 85 manually designed attributes [16]. The AwA dataset is usually used to evaluate compact representation learning methods [7, 26] for image

classification tasks. The second dataset is the ILSVRC2010, a popular dataset for benchmarking large scale image classification, which contains 1.2 million images from 1,000 diverse categories [6].

Experiment Settings We conducted experiments on the above datasets under following two different settings, in order to study various aspects of ibpCNN.

Fully supervised learning. Here we study the benefits brought by the compactness of ibpCNN for image classification tasks, and investigate how the structures of ibpCNN adapts to varying data complexities. For both AwA and ILSVRC2010 datasets, we split them into training, validation and test sets following the conventions in previous works [16, 6]. To simulate varying data complexities, we use different amounts of training and validation images ($10, 20, 30, 50, 100\%$ of the available ones respectively) for training, and evaluate the methods on the raw test sets.

Semi-supervised learning. Here we target at studying the ability of ibpCNN in exploiting unlabeled data to learn complicated data distributions, with rather limited supervision information. We split the training sets of the AwA and ILSVRC2010 into labeled and unlabeled subsets. The sizes of labeled subset are fixed as 30% (for AwA) and 10% (for ILSVRC2010) of the original training set respectively. We vary the size of unlabeled subsets, such that the ratio of # unlabeled to # labeled images increases from $1, 5, 10$ to 50. To collect enough unlabeled images, extra images are randomly sampled from ImageNet [6] without having overlapped categories with labeled images. We evaluate the image classification performances on the original test sets.

Baselines We compare ibpCNN with both classical hand-crafted feature based image classification methods and deep learning methods. In particular, the hand-crafted feature based methods include (1) the hand-crafted low-level features as used in [16]; (2) the category-level discriminative attributes of [26]; (3) classemes [25]; and (4) the attributes from dictionary learning method [7]. All these baseline methods (except for (1)) aim at learning compact image representations from low-level hand-crafted features. A linear SVM based classifier is applied on the output representations from these baseline methods, whose penalty parameter C is tuned on the validation sets.

As for the deep learning baselines, we mainly compare with CNN model of AlexNet architecture [15] in the fully supervised learning setting, which has shown state-of-the-art performance on several benchmark datasets in previous studies. In the semi-supervised learning setting, in addition to CNN-AlexNet, we also compare ibpCNN with Conovlutional Auto-Encoder (CAE) [19], which is capable of modeling data generation in an unsupervised manner without end-to-end training. We also report the performance of CNN and CAE models with the architecture identified by the GAP algorithm, *i.e.*, the CNN-GAP and CAE-GAP models. A common practice is to initialize parameters of CNN models with CAE models [19], denoted as CAE+CNN. We also compare ibpCNN with this baseline model in the semi-supervised learning setting. For fair comparisons, the layer numbers of CNN and CAE models are set the same as ibpCNN's and the architecture of CAE exactly follows the one specified in [19]. All the deep neural network models are trained from scratch.

CNN-AlexNet and low-level features cannot directly perform semi-supervised learning. We apply a graph based semi-supervised learning algorithm proposed in [23] to propagate the annotations from labeled data to unlabeled data based on their output representations. We then re-train the models (CNN-AlexNet and linear SVM) on the augmented data. As CAE cannot be trained end-to-end for classification loss directly, we train a linear SVM model on the CAE output representations, whose parameter is tuned via cross-validation on the validation set.

Implementation Details On the AwA dataset, we use 4 convolutional layers and 1 fully connected layers for ibpCNN. Each convolutional layer is followed by a pooling layer. On ILSVRC2010 dataset, since the number of training images is much larger compared with AwA, we use more layers for ibpCNN, which consist of 5 convolutional layers and 2 fully connected layers. At the top layer of the networks, a softmax combined with cross-entropy loss is applied. All the images are pre-processed in the Caffe framework [13]. The trade-off parameter $\lambda^{(\ell)}$ in ibpCNN is set as $0.1/N^{(\ell)}$ across all the layers, where $N^{(\ell)} = 1 \times 10^5/2^\ell$ is the allowed maximum number of input feature maps for the ℓ-th layer (see Algorithm 1). The size upper bounds for the layers from bottom to top in ibpCNN exponentially decrease as top layers extract visual patterns at higher abstract level than bottom layers and thus need less filters. As long as we specify a sufficiently large upper bound on the layer size (*e.g.*, 1×10^5 in this work), the GAP algorithm provably provides an optimal architecture eventually and we empirically found its results are rather *robust* to the value of trade-off parameters λ.

6.2. Results and Discussions

6.2.1 Fully Supervised Learning

We first present image classification results of ibpCNN and baseline methods, under the fully supervised learning setting. We also demonstrate how the structure of ibpCNN adapts to the data complexity, and report the computation costs of ibpCNN for training and testing.

AwA Dataset The classification the AwA dataset of ibpCNN and baselines, with varying sizes of training data, are plotted in the left panel of Figure 3. The results offer following observations. First, the ibpCNN model provides

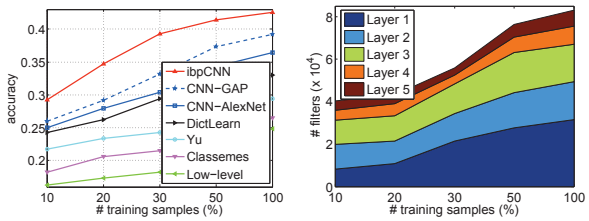

Figure 3. Results on AwA dataset. Left: comparisons between ibpCNN and baselines with different numbers of training images. CNN-GAP is the CNN model with the architecture determined by GAP. Right: learned number of filters per layer in ibpCNN trained with different numbers of training images (best viewed in color).

a significant performance enhancement over baselines. For instance, when using 30% of the training data, ibpCNN outperforms the best baseline (CNN-AlexNet) with a margin of nearly 10%. In fact, with only 10% of the training samples, ibpCNN has already achieved comparable performance than CNN-AlexNet with 30% of the training data. These results clearly demonstrate the power of ibpCNN in adapting model to data complexity and providing better generalization performance. Secondly, increasing the amounts of training data from 10% to 30% improves the performance of ibpCNN more significantly than CNN-AlexNet and other baselines. This also demonstrates that when the training data are limited, the adaptive structure enables ibpCNN to benefit more from added training data, compared with models of fixed structures. Thirdly, CNN-AlexNet – as a deep model – does not perform significantly better than hand-crafted feature based methods (the DictLearn), when the training samples are scarce (30% or less). This is because the number of parameters in CNN-AlexNet is too huge and the provided training data are not sufficient for optimizing the parameters. This also validates the motivation of this work to pursue the balance between model complexity and data complexity. Employing the architecture identified by GAP boosts the performance of CNN (CNN-GAP vs. CNN-AlexNet). However, CNN-GAP is still inferior to ibpCNN which jointly optimizes the data modeling and task fitting.

Learned Structures of ibpCNN Figure 3 (right) shows how the learned structures of ibpCNN varies with the size of training data. The results confirm the adaptivity of ibpCNN: the sizes of its learned layers – and thus its model complexity – increase monotonically with the data complexity. More specifically, we can observe that when the amount of training data increases from 20% to 50%, the size of ibpCNN grows very rapidly. Note that the fully connected layer (layer 5) always stays at a relatively small size (around 1,500 filters), which indicates the output representations of ibpCNN have low-dimensionality and are compact. Convolutional layers are much larger than fully connected layers.

Table 1. Model size of ibpCNN in comparison with CNN-AlexNet, and speed-up brought by ibpCNN over CNN-AlexNet for test image prediction. Results are obtained on the AwA dataset.

# training	10%	20%	30%	50%	100%
# filters	6.4%	6.7%	7.8%	10.9%	12.5%
speed-up (rel. CNN-AlexNet)	11×	10×	9×	7×	6×

These results are consistent with our intuitions: more complicated data usually demand more sophisticated model for reliable distribution modeling, and require more low-level filters for extracting primitive edge and corner patterns.

Computation Costs It takes ibpCNN 0.72 and 2.67 hours for learning the structure and optimizing parameters respectively. The comparisons between ibpCNN and CNN-AlexNet in terms of their computation cost are reported in Table 1, where we also report the relative size of ibpCNN w.r.t. CNN-AlexNet. In the training phase, structure learning consumes much less computation time than parameter learning (including tuning filters and inferring representations), which demonstrates the GAP algorithm for structure learning is efficient and brings minor computation overhead. Table 1 focuses on comparing efficiency of ibpCNN with CNN-AlexNet model for forward passing. As can be observed, ibpCNN is much smaller than CNN-AlexNet in terms of the number of filters. Benefited from such compact structure, ibpCNN is significantly more efficient than CNN-AlexNet for classifying new images. For fair comparison, both ibpCNN and CNN-AlexNet are tested using CPU mode without any engineering acceleration.

ILSVRC2010 Dataset Table 2 shows the multi-class classification accuracy of different methods, using varying amounts of training images, on the ILSVRC2010 dataset. The ibpCNN outperforms CNN-AlexNet (which was ever the most successful deep network architecture on ILSVRC2010) and other baselines significantly when the training samples are scarce. For instance, when only using 10% of the training data, ibpCNN outperforms CNN-AlexNet by nearly 11%. The performance gain is based the appropriate complexity of ibpCNN which has only 17.5% of filters in CNN-AlexNet. When the training samples are sufficient, the performance ibpCNN is slightly inferior to CNN-AlexNet. However, ibpCNN has much a smaller size than CNN-AlexNet (only 37.5% filters of CNN-AlexNet) and thus it costs much less time for testing. This is a quite appealing feature in practice. Here for fair comparison, both CNN-AlexNet and ibpCNN are trained from scratch, and only a single model is tested. Comparing CNN-GAP with CNN-AlexNet and ibpCNN offers following observations: CNN-GAP outperforms CNN-AlexNet significantly when the number of training examples is very small (e.g., when less than 10% of training examples are used). This clearly demonstrates the ability of the proposed GAP algorithm in offering more compact model architecture when

Table 2. Comparisons of top-1 classification accuracy on ILSVRC2010, with different numbers of training images (in percentages of the available training images). Both CNN-AlexNet and ibpCNN are trained from scratch. The number of filters of ibpCNN is also reported in comparison with CNN-AlexNet.

# training	1%	5%	10%	50%	100%
DictLearn [7]	14.23	18.68	21.38	29.71	31.24
CNN-AlexNet [6]	5.02	10.80	21.17	**47.12**	**61.7**
CNN-GAP	11.23	15.91	26.21	41.23	54.38
Ours	**17.79**	**23.36**	**31.04**	46.87	58.07
# filters (rel.)	4.2%	14.5%	17.5%	28.2%	37.5%

training examples are scarce and better performance. Secondly, ibpCNN consistently outperforms CNN-GAP due to that ibpCNN also models data distribution through reducing the data reconstruction loss. Such an additional generative component to the classification loss further enhances the performance of ibpCNN.

6.2.2 Semi-supervised Learning

We here experimentally evaluate the performance of ibpCNN for semin-supervised learning. The details of how to apply CNN-AlexNet and low-level features in semi-supervised learning are provided in the baseline descriptions. The comparison results of ibpCNN with other baselines are plotted in the left panel of Figure 4. The results confirm that ibpCNN consistently outperforms the baselines with varying numbers of training samples. We can observe that ibpCNN benefits more from the unlabeled data in achieving better generalization performance (higher testing accuracy) than CAE, CAE-GAP and CAE+CNN+GAP models, which demonstrates ibpCNN is better at modeling the joint data distribution through jointly optimizing the model structure, parameters and data representations. Comparing CAE-GAP with CAE confirms that the GAP algorithm can also provide better structure for generative models. The observation that CAE+CNN+GAP outperforms both CAE-GAP and CNN-GAP verifies the advantages of combining discriminative and generative components in ibpCNN for the semi-supervised learning tasks. Similar to the supervised learning setting, we also investigate the learned structure of ibpCNN with different amounts of training data. The adaptive structures of ibpCNN are shown in Figure 4 (right). More complicated data (including more unlabeled data) results in a larger ibpCNN, which verify the adaptivity of ibpCNN's structure.

Results on ILSVRC2010 are presented in Table 3. Convolutional Auto-Encoder (CAE) [19] and ibpCNN are among the best performing methods, as they are able to model data distribution without requiring supervision information. Initializing the parameters of CNN with CAE and further fine-tuning the CNN on labeled training examples offers better performance than single CNN and CAE. This verifies the effectiveness of adding additional generative components in the overall loss as in ibpCNN in utilizing the large amount of unlabeled examples. The ibpCNN model consistently outperforms the baselines, in particular CNN+CAE+GAP, with varying numbers of external unlabeled data. This demonstrates the advantages of its adaptive structure to data complexity as well as its joint end-to-end learning strategy for adapting the model structure, inferring the representations and optimizing the model parameters.

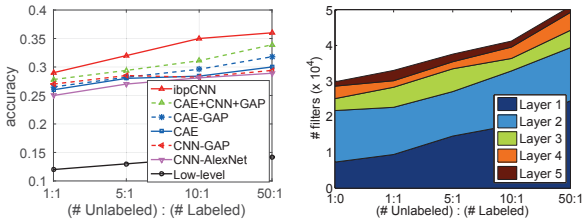

Figure 4. Comparisons of different methods on the AwA dataset, under semi-supervised learning setting. Left: Average classification accuracies of various methods. CAE-GAP, CNN-GAP and CAE+CNN+GAP employ the architectures determined by the GAP algorithm, and CAE+CNN+GAP is the CNN model whose parameters are initialized by CAE. Right: number of learned filters per layer in ibpCNN. Here, 1:0 on x-axis means only using 30% of unlabeled data (best viewed in color).

Table 3. Comparisons of top-1 classification accuracies of different methods on ILSVRC2010 dataset, under semi-supervised learning setting. The number of labeled images is fixed as 10% of the available ones, and the number of unlabeled images varies.

Unlabeled : Labeled	1 : 1	5 : 1	10 : 1	50 : 1
Low-level feature [16]	15.42	16.17	16.92	17.67
CNN-AlexNet [6]	21.92	23.19	23.23	24.42
CNN-GAP	24.32	25.23	26.98	27.39
CAE [19]	26.70	28.06	30.64	31.45
CAE-GAP	27.92	29.20	32.78	33.81
CAE+CNN+GAP	29.42	31.93	33.23	35.79
ibpCNN (ours)	**32.42**	**33.01**	**34.36**	**37.70**

7. Conclusions

We proposed a novel deep model, ibpCNN, whose structure is automatically adaptive to both the training data complexity and specified supervised task loss function(s). Our flexible ibpCNN model was effectively applied to image classification tasks and produced compact and scalable image representations. We demonstrated the superior performance and efficiency of ibpCNN compared with standard hand-crafted CNN models.

Acknowledgment

This work was supported by DARPA, AFRL, DoD MURI award N000141110688, NSF awards IIS-1427425 and IIS-1212798, and the Berkeley Vision and Learning Center.

References

[1] R. P. Adams, H. M. Wallach, and Z. Ghahramani. Learning the structure of deep sparse graphical models. *arXiv preprint arXiv:1001.0160*, 2009.

[2] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al. Greedy layer-wise training of deep networks. 2007.

[3] T. Broderick, B. Kulis, and M. Jordan. MAD-Bayes: MAP-based asymptotic derivations from Bayes. In *ICML*, 2013.

[4] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *FOCS*, 2012.

[5] J. Chung, D. Lee, Y. Seo, and C. D. Yoo. Deep attribute networks. *arXiv preprint arXiv:1211.2881*, 2012.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[7] J. Feng, S. Jegelka, S. Yan, and T. Darrell. Learning scalable discriminative dictionary with sample relatedness. In *CVPR*, 2014.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv preprint arXiv:1311.2524*, 2013.

[9] T. Griffiths and Z. Ghahramani. Infinite latent feature models and the Indian Buffet Process. In *NIPS*, 2006.

[10] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Simultaneous detection and segmentation. In *ECCV*. 2014.

[11] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[12] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[14] K. Kavukcuoglu, M. Ranzato, and Y. LeCun. Fast inference in sparse coding algorithms with applications to object recognition. *arXiv preprint arXiv:1010.3467*, 2010.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[16] C. Lampert, H. Nickisch, and S. Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *CVPR*, 2009.

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

[18] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In *NIPs*, volume 2, pages 598–605, 1989.

[19] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Artificial Neural Networks and Machine Learning–ICANN 2011*, pages 52–59. Springer, 2011.

[20] Y. Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[21] J. Paisley and L. Carin. Nonparametric factor analysis with beta process priors. In *ICML*, 2009.

[22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[23] A. Subramanya and J. A. Bilmes. Entropic graph regularization in non-parametric semi-supervised classification. In *Advances in Neural Information Processing Systems*, pages 1803–1811, 2009.

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[25] L. Torresani, M. Szummer, and A. Fitzgibbon. Efficient object category recognition using classemes. In *ECCV*, 2010.

[26] F. Yu, L. Cao, R. Feris, J. Smith, and S.-F. Chang. Designing category-level attributes for discriminative visual recognition. In *CVPR*, 2013.

[27] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010.

[28] M. D. Zeiler, G. W. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2018–2025. IEEE, 2011.

[29] G. Zhou, K. Sohn, and H. Lee. Online incremental feature learning with denoising autoencoders. In *AISTATS*, 2012.