CONVOLUTIONAL RESTRICTED BOLTZMANN MACHINES FOR FEATURE LEARNING

by

Mohammad Norouzi
B.Sc., Sharif University of Technology, 2007

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Fall 2009

APPROVAL

Name: Mohammad Norouzi

Degree: Master of Science

Title of Thesis: Convolutional Restricted Boltzmann Machines for Feature Learning

Examining Committee: Dr. Andrei A. Bulatov
Chair

____
Dr. Greg Mori, Senior Supervisor

____
Dr. Robert F. Hadley, Supervisor

____
Dr. Anoop Sarkar, SFU Examiner

Date Approved: Nov 27, 2009

Declaration of Partial Copyright Licence

Abstract

In this thesis, we present a probabilistic generative approach for learning hierarchical structures of spatially local features, effective for visual recognition. Recently, a greedy layerwise learning mechanism has been proposed for training fully-connected neural networks. This mechanism views each of the network's layers as a Restricted Boltzmann Machines (RBM), and trains them separately and bottom-up. We develop Convolutional RBM (CRBM), in which connections are local and weights are shared to respect the spatial structure of images. We train a hierarchy of visual feature detectors in layerwise manner by switching between the CRBM models and down-sampling layers. Our model learns generic gradient features at the bottom layers and class-specific features in the top levels. It is experimentally demonstrated that the features automatically learned by our algorithm are effective for visual recognition tasks, by using them to obtain performance comparable to the state-of-the-art on handwritten digit classification and pedestrian detection.

Acknowledgments

This is a great opportunity for me to publicly thank those who have been influential during my studies at Simon Fraser University. I am grateful to God who gave me this wonderful life and continuously showers me with blessings. I am profoundly indebted to my supervisor, Dr. Greg Mori, because of his calmness, endless support, and deep insights. Whenever I was disappointed of the ideas or experimental results, he inspired me to try harder. I thank my Mom and Dad, Mansoureh and Sadegh, for their unconditional love and support even from oversees. I would like to thank Dr. Bob Hadley who gave me some invaluable understanding of cognitive science and neural networks, and Dr. Anoop Sarkar who graciously introduced me to graduate research and machine learning. I was very fortunate to be a member of vision and media lab at SFU. I kindly acknowledge all of my fellow students for thier ideas and support. Especially I should thank Mani Ranjbar for his close cooperation and helpful discussions. Finally, I warmly thank all of my friends and previous teachers.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The dream of building artificially intelligent agents is tightly coupled to developing reliable visual processing systems. The importance of vision in communicating with the world and in shaping our thoughts is inevitable. During the last few decades, researchers from different discipline perspectives such as computer vision, machine learning, cognitive science, neuroscience, etc. have tackled the problem of design and analysis of visual perception models, and remarkable successes have been achieved. However, there is still a long way to go.

In any visual processing model, a set of features is extracted from the input and further processes are carried out on the feature representation, instead of the plain input. In our view, features are statistical regularities inherent in the input, and for vision applications we are seeking features that are informative, robust to changes in illumination, and invariant to shifts and distortions of the input image. In this work, we focus on learning hierarchical feature representations of images. The Use of a hierarchical structures of features is mainly inspired by allusions to the biological processing systems such as human's brain. In such representations, features are placed in a multilayer pyramid and are built on top of each other. As we move upward in these hierarchies features become larger, more complicated, and less frequent. For example, an upper layer feature might symbolize an object part or even a full object, while a lower level feature might just be an oriented edge segment.

Undoubtedly, building hierarchical features is quite useful for a lot of visual processing applications. However, a learning algorithm that is scalable, generalizes well, and benefits from un-labeled data is lacking. This work is a step forward towards learning better, larger, and deeper feature structures with less amount of (labeled) data.

The thesis is organized as fallows. The reminder of introduction describes the problem of object detection as the testbed of our features, and next, we sketch an overview of our feature detector hierarchy and the proposed learning method. Chapter 2 discusses the huge amount of related work in this area. Chapter 3 is dedicated to some background knowledge about the models and learning algorithms that we build upon. Chapter 4 gives a detailed description of our model and the learning algorithm. Chapter 5 presents experimental evaluation and Chapter 6 concludes the thesis.

1.1 Object Detection

Object detection in general is the problem of detecting and localizing objects of certain categories (such as humans or cars) in images and videos. This problem is often simplified as finding rectangular bounding boxes that surround objects of interest in still images. For instance, current digital cameras solve object detection for faces and draw bounding boxes around them. To further simplify the detection problem, the so-called *sliding window approach* is popular. In this approach, a fixed size window is slid all over the image, and at each window location a binary decision is made to determine whether the object is present in that bounding box or not. To handle changes in the object size, the input image is scaled at certain ratios and the same naive procedure of window sliding is repeated. Thus, in this framework the detection problem reduces to a binary classification, where the inputs are image subwindows scaled to the same size. Then the classifier should decide whether the object is centered and bounded within a given subwindow or not.

In our model for object detection we follow the sliding window approach. This lets us mainly focus on the feature learning procedure, and do experiments on different aspects of the learning without worrying about object localization. Although classification and localization are not completely separable, this is a helpful simplification. For the final classification we use an off-the-shelf method i.e., Support Vector Machine (SVM) over the induced features, so we do not have to worry about the classification component either.

1.1.1 Pedestrian Detection

As a specific instance of object detection, we are interested in the challenging task of *pedestrian detection* in still images. By pedestrians we refer to humans that are almost fully visible and standing roughly upright. The difficulty of pedestrian detection is due to humans' diverse appearance and the wide range of poses that humans can take. As always, background clutter and changes in illumination are the other issues that should be addressed. However, since people move a lot and are the most important subject of images, these problems become more severe for the task of human detection.

It is worth mentioning that pedestrian detection has quite a number of applications in intelligent vehicles, security, robotics, human computer interaction, and image search and retrieval. Improving the safety of pedestrians by installing pedestrian detection modules in automobiles is one of the most important applications of this research, and currently car manufacturers are doing and funding research in this field. We perform our experiments on a human dataset called INRIA [7] which was built from personal web albums.

It is important to note that our system is not particularly designed for pedestrian detection, and we only treat this task as an evaluation criteria to measure how well our feature learning approach performs.

1.1.2 Handwritten Digit Classification

Besides pedestrian detection, we do some experiments on handwritten digit classification. A standard dataset called MNIST [1] is available for this task that contains thousands of cropped and centered handwritten digits for training and test. This dataset is attractive for us because many approaches have been evaluated on it, and we can compare our model with them without any cost. Further, the simple setup of MNIST digits provides a framework for adjusting some of the details of our model and the learning algorithm.

1.2 Features

The success or failure of an object recognition algorithm hinges on the features used. Successful algorithms have been built on top of hand-crafted gradient response features such as SIFT [24] and histograms of oriented gradients (HOG) [7]. While their successes have been demonstrated in a variety of domains, they are fixed features that cannot adapt to model the intricacies of a particular problem. A competing approach, followed in this work, is to automatically learn spatially local features tuned for a particular visual recognition task. In this work, by learning features, we mean learning a set of feature representatives or *filter kernels* that could be used in a filtering procedure to induce actual features.

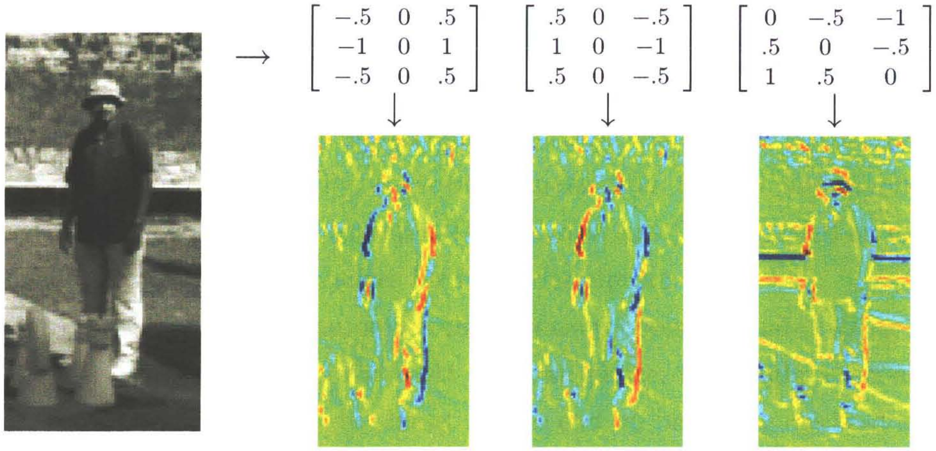

$\begin{bmatrix} -.5 & 0 & .5 \\ -1 & 0 & 1 \\ -.5 & 0 & .5 \end{bmatrix}$
$\begin{bmatrix} .5 & 0 & -.5 \\ 1 & 0 & -1 \\ .5 & 0 & -.5 \end{bmatrix}$
$\begin{bmatrix} 0 & -.5 & -1 \\ .5 & 0 & -.5 \\ 1 & .5 & 0 \end{bmatrix}$

Figure 1.1: Illustration of filtering operation. A 128×64 pedestrian image (left) is filtered with 3×3 filter kernels (shown above), and the filter responses are visualized (bottom). In the responses hot colors (reddish) represent positive large values and cold colors (bluish) denote negative large values.

A standard operation for extracting spatially local features from images is called *filtering*. A filter kernel (or filter for short) is a rectangular matrix of coefficients that represents a spatially local visual feature (see Fig. 1.1). Filters are typically much smaller than the input images. To extract responses of an image to a filter, We carry the given filter over all neighborhoods of the image, and extract feature responses from different locations. The operation of computing the dot product of a vectorized filter and different subwindows of the the input is called filtering. We explained the filtering operation in more detail in Appendix A.1. The goal of our work is to learn the filter parameters from a set of training examples. In this thesis, we sometimes refer to filters as features and we use filter responses and feature responses interchangeably to refer to actual induced features.

In Fig. 1.1 a pedestrian image is filtered with 3×3 filter kernels, and the filter responses are visualized. Note that the 3×3 filters of Fig. 1.1 act as oriented edge detectors because they subtract the intensity of nearby pixels in specific directions. These type of filters are called oriented edge or gradient filters. Often classification on the gradient responses performs much better than building a classifier on pixel values.

Widely used image descriptors such as SIFT [24] and HOG [7] are cascades of few components. First, oriented edge features are extracted from local regions of the input in a filtering step. Next, edge responses are passed through a nonlinear function such as absolute value or soft thresholding. Then, a pooling operation follows that aggregates responses over regions of the image and reduces the dimensionality. Finally, the outputs are locally contrast normalized. In the design of hand-crafted descriptors, the filter parameters, as well as the choice of non-linearity, contrast normalization, the size and number of filters need to be specified carefully. For example, Dalal and Triggs [7] conducted an extensive set of experiments to make the best choices for HOG descriptors.

The theme of this work is to automate the tedious process of hand-crafting features by learning the filter coefficients from data. At the first glance, this might seem unnecessary, since the low-level filters surprisingly almost always end up becoming oriented edges; when they are statistically learned from training images [28], when filters are tuned to fit the visual cortex data [16, 34], and when they are crafted by hand [24, 7]. Although learning filters is still theoretically and biologically appealing, and even fruitful for other applications, it is not very interesting for natural images since we know how the filters would look like in advance. As it is expected, the filters learned in our experiments for natural images also resemble the oriented edge detectors, which are shown in Fig. 5.4.

However, consider the image patches in Fig. 1.2. Each plate in this figure includes a set of patches from pedestrian images that highly activated a feature automatically learned by our algorithm. The algorithm learns individual features corresponding to areas around the head, feet, and inverted-"V" patterns around the legs. Unlike low-level filters, hand-crafting features such as these would be impossible. These features are generally larger and have more free parameters than the gradient filters. Further, they are highly task-specific. These characteristics highlight the essence of employing statistical learning approaches for inducing such large scale features. In our experiments, we demonstrate that combining the learned task-specific features with the generic HOG gradient responses leads to state-of-the-art performance on the challenging INRIA pedestrian detection benchmark.

1.3 Local Feature Detector Hierarchies

So far, we mentioned low-level (gradient) and high level class-specific features, and argued that inducing higher level regularities requires a learning procedure. But, we have not discussed how these two could be related in a model. Our model is a multilayer network, in which larger scale features are built on top of the lower level attributes. This leads to a

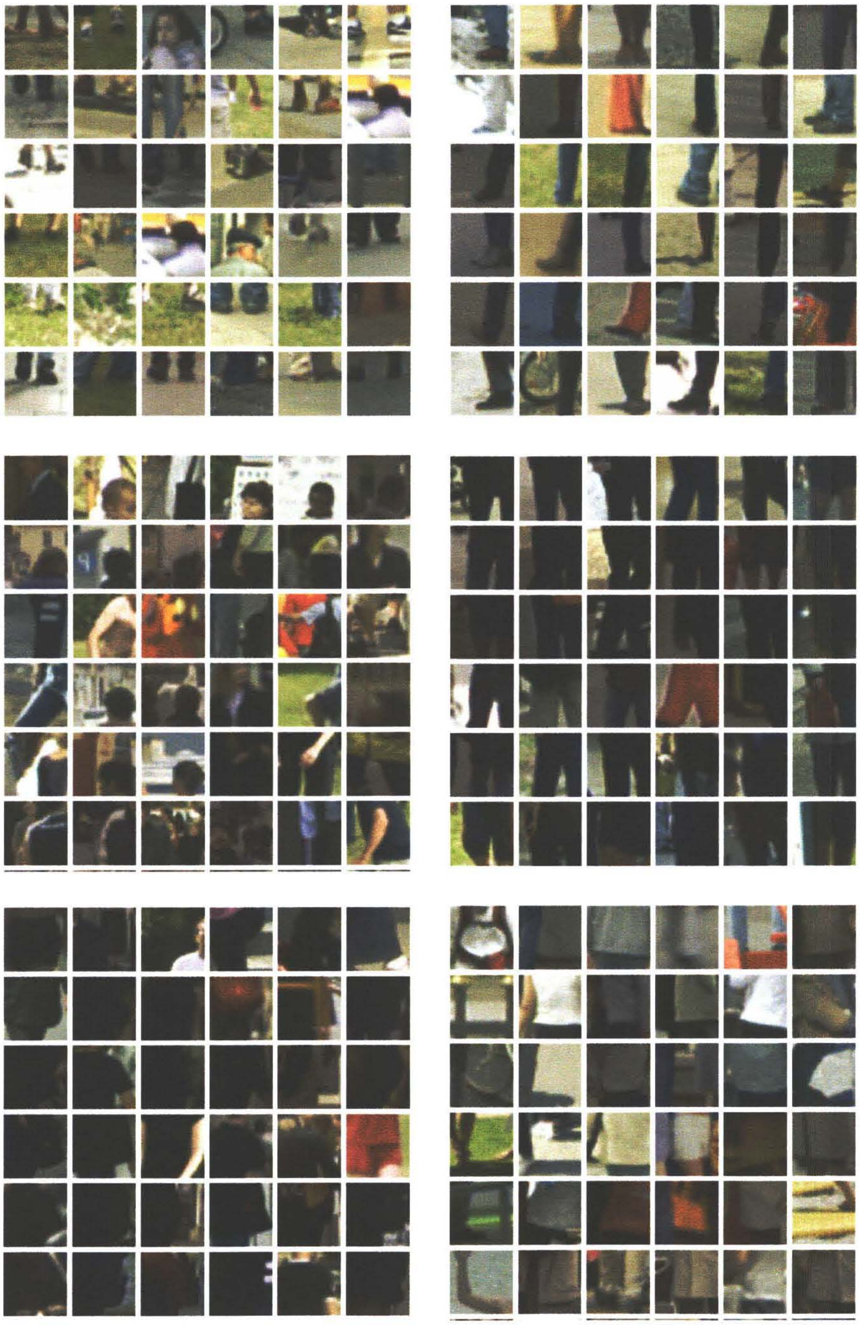


Figure 1.2: Large scale features learned by the proposed model from pedestrian images. Each plate corresponds to a set of patches that highly responded to a feature.

feed-forward neural network that consists of two stages of feature extraction. Each feature extraction stage itself is a sequence of filtering, non-linearity, and pooling operations. The output of the first stage feature extractor is given as input to the second stage. This multi-stage feature detector is visualized schematically in Fig. 1.3 as a multilayer network of alternating filtering/non-linearity and pooling layers. Note that contrast normalization is a component of general feature extractors, but is removed from our current model to make it simpler. Although we limited our model to only two stages of features, there is no reason that this model could not be extended to deeper architectures. This is specifically due to the learning algorithm that we propose. This learning trains the model one layer at a time, so it scales well with the number of layers in the structure.

The idea of building hierarchical structures of features for object detection has deep roots in the computer vision literature, with the development of many such models inspired by allusions to the human visual system [16, 9, 21, 34, 25, 30]. We generally refer to these models as *local feature detector hierarchies (LFDH).* The basic aspect of such hierarchies is that feature detectors activate if a familiar pattern was present among the lower level features connected to them. This enables the model to extract primitive and generic attributes at the bottom layers, and put them together to compose more complicated and specific features at the upper layers. Another important property of these models, which makes them suitable for vision problems, is that their connections are local. In other words, each feature is connected to only a small subset of features in the layer below it. Local connectivity reduces the number of parameters and respects the spatial structure of images. Thus, the network is designed such that each feature detector would be responsible for patterns in a local region of the input image. Note that as we move upward in the hierarchy the features become larger and the image regions that affect their activity grow.

Here we describe components of local feature detector hierarchies in more detail. We should distinguish between these models, which are fine-tuned for vision applications, and generic multilayer fully connected networks. This difference especially is important for us because our contribution lies in adopting Hinton's layerwise learning algorithm of generic networks for locally connected structures. We should acknowledge that our general definition of LFDH is highly influenced by a recent work of Jarrett et al. [17]. As sketched above, a LFDH is a stack of multiple stages of feature detection, where each feature detection consists of filtering, non-linearity, pooling, and normalization steps (Fig. 1.3). Not necessarily all of the LFDH-based models have all of these modules, but these are the typical operations.

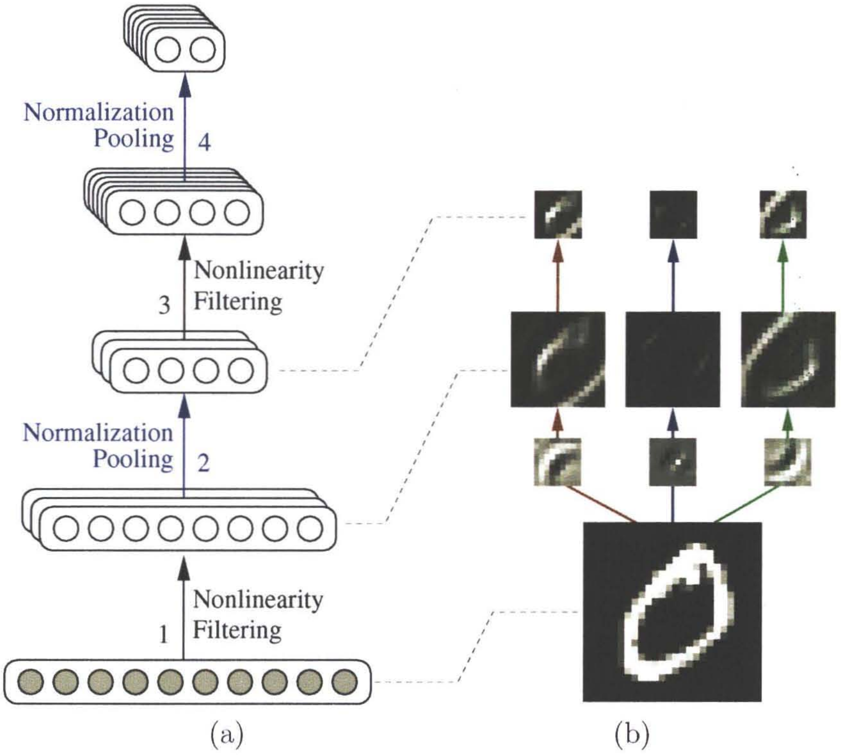


Figure 1.3: **(a)** Illustration of general hierarchical feature detectors. A two stage feature detector is visualized. Each feature is extracted via a pipeline of filtering, non-linearity, pooling, and normalization. **(b)** First two layers of the hierarchy. Filtering and 2×2 maximum pooling operations are shown performed on a handwritten digit example.

Filtering. The first step in feature extraction is usually filtering. We described the filtering operation above. This step provides us with a matrix of responses representing features at different locations. Clearly, we need a set of distinct filters to extract different types of features. In Fig. 1.3b filtering and its subsequent pooling are illustrated.

Non-linearity. Filter responses are not in a specific range. They might be positive, negative, large, or small depending on the values of the input and coefficients of the filters. Normalizing the filter responses is often beneficial. For doing so, a nonlinear transformation becomes handy. We use the logistic sigmoid function $f(x) = 1/(1+\exp(x))$ to map the filter responses between zero and one. It is noteworthy that we did not choose sigmoid non-linearity a-priori, instead it arose naturally due to the probabilistic interpretation of binary features in our model.

Pooling. Another important component of LFDHs is the pooling layer (also referred to as subsampling or down-sampling layer). This layer aggregates its inputs over regions of the image, and reduces the dimensionality of the feature representation. Besides dimensionality reduction, the pooling operation makes the representation robust against small shifts and distortions of the input.

In our implementaion we coducted down-sampling by performing *max pooling*. In max pooling the input matrix is devided into a grid of non-overlapping sub-matrices, and the entries in each subwindow are substituted by their maximum value. For example for performing a 2×2 max pooling, we divide the input matrix into non-overlapping 2×2 sub-matrices. Then we replace the entries in each sub-matrix with a single value: the maximum value of that sub-matrix. Therefore by 2x2 max pooling the dimensionality of the input is reduced by a factor of 4 (see Fig. 1.3b for illustration). It is noteworthy that some other types of down-sampling operations such as (weighted) average pooling, probabilistic max pooling [23], etc. have similar characteristics as the max pooling described above. A quantitative example of max pooling is given in Appendix A.2.

Normalization. Local contrast normalization at the end of each feature detection is motivated by computational neuroscience models [29]. This operation especially helps the features to become robust to changes in illumination condition. The HOG descriptor implements contrast normalization in overlapping blocks of aggregated filter responses, and it is known that HOG's success is highly influenced by this operation [7]. In our current model we did not implement contrast normalization, mainly because it adds another complexity to the model. However, for human detection task, we combined HOG descriptors with our high level features and this combination gives us the desired illumination invariance to some extent.

In summary, when an input image is provided to our model, it is filtered using a first filter bank, then the responses are passed through a sigmoid non-linearity, followed by max pooling in local neighborhoods. Next, second filter bank, sigmoid, and max pooling layers come into action and produce the final outputs. We refer to these final responses as the high level feature responses, and use them for classification.

1.4 Learning

Even though the idea of local feature hierarchies sounds very promising for object detection and image modeling, a learning algorithm that is scalable, fast enough, and benefits from unlabeled data is lacking. On the other hand, neuroscientists believe that a similar sequence of filtering and pooling layers (resembling simple and complex cells) is being employed within the visual cortex of mammals [16, 4], and a biologically implementable algorithm for learning such networks would be of their interest as well. This work is a step forward towards learning larger and deeper hierarchical feature detectors with less amount of (labeled) data. Also, because we train the model unsupervised, layerwise, and only use the local signals for training each layer, it is possible that the brain is executing a learning algorithm with the same flavor.

1.4.1 Backpropagation Algorithm

A well studied algorithm for tuning parameters (weights) of multilayer neural networks is known as Backpropagation (BP) [33]. Fully connected networks and LFDHs can both be learned using backpropagation. This algorithm drives a neural network to mimic a function that maps the input data into a desired output e.g., correct labels. For a set of input/output pairs, the parameters of the network are iteratively updated to minimize the discrepancy between the network's outcomes and the correct outputs. For doing so, a cost or loss function is defined to measure how well the network performs, and backpropagation provides an efficient way for propagating the gradient of this loss function into the parameters. During the learning, the network parameters are updated by gradient descent to minimize this loss function.

Backpropagation (BP) can be used for supervised or unsupervised learning. In supervised learning, we have a number of training examples, and their corresponding labels are available. A network is constructed with a number of hidden layers to map the data into its true label (e.g. BP for handwritten digit classification [21]). Note that the hidden layers represent adaptable latent features. The weights of a multilayer network would be randomly initialized and iteratively updated to reach a (stable) local minima of the loss function.

In unsupervised learning, training data are un-labeled, and the goal could be density estimation, extracting regularities (features) inherent in the data, or dimensionality reduction. Again, BP can be exploited by training a family of multilayer networks called *autoencoders.* An autoencoder is a model that has identical input and output layers, and is designed to reconstruct the original data from an intermediate representation. Autoencoders might be employed for inducing low-dimensional codes or high-dimensional sparse representations. In the latter case, the idea is that a set of features that can reconstruct the data well and have certain other characteristics such as sparsity and high dimensionality can provide a useful representation for other tasks such as classification.

1.4.2 Shortcomings of Backpropagation

There are two main issues involved with the BP algorithm, especially when we want to train deep structures. Hinton describes the first problem in [15] as

> It is necessary to choose initial random values for all the weights. If these values are small, it is very difficult to learn deep networks because the gradients decrease multiplicatively as we backpropagate through each hidden layer. If the initial values are large, we have randomly chosen a particular region of the weight-space and we may well become trapped in a poor local optimum within this region.

Second, when labeled data are limited and the number of free parameters in the network is large, BP performs poorly. This is a case of overfitting and is mentioned as overparametrization in [30]. Although some methods such as L2 regularization of weights or early stopping have been proposed to tackle overfitting, still BP on large and deep networks does not perform well.

1.4.3 Greedy Layerwise Learning

To address the shortcomings of BP, Hinton et al. [13] proposed a learning procedure known as greedy layerwise learning for fully connected networks. In this mechanism layers of a generic neural network are trained one layer at a time and from bottom layer upward. When layers are trained separately multiplicative reduction of gradients does not occur because only one layer is involved. Also, over-parametrization is not an issue because each layer has a much smaller number of weights than the whole network. After completion of layerwise training, BP can be adopted to fine-tune the parameters regarding the feedback of higher layer weights.

Training the bottom layers of a network without considering the upper levels cannot be done in supervised fashion. Instead, a probabilistic model called Restricted Boltzmann Machine (RBM) [35] that resembles a two-layer fully connected network was developed for unsupervised learning of each network's layers. The RBM is a probabilistic model for a density over observed variables (e.g., over pixels from images of an object) that uses a set of hidden variables (representing presence of features). The intuition behind unsupervised learning of the RBMs is to extract a set of features that are able to model the training data density and can reconstruct the inputs well. Layers of a network are trained without considering the labels and viewed as separate RBMs from the bottom layer upward. After each RBM is trained, its weights are frozen and it is stacked on top of the previously learned layers. The input goes through the frozen layers and reaches a singled out layer that should be trained. When we reach the topmost layer, labels become available and a classifier would be trained on the extracted features. It has been shown that a network which is pre-trained in this way performs much better than a network trained using conventional BP over random initialization [14, 13].

1.5 Contribution

We develop a layerwise learning algorithm for locally connected feature hierarchies. The algorithm that we develop is highly influenced by Hinton's greedy layerwise learning [13] for fully connected networks. As noted above, greedy layerwise learning is based on the RBM model. In the standard RBM all observed variables are related to all hidden variables by different parameters. While this model can be used to create features describing image patches, it does not explicitly capture the spatial structure of images. Instead, we incorporate ideas from the convolutional neural network (CNN) of LeCun et al. [21] and develop a model called *Convolutional RBM* (CRBM). We define patterns of weight sharing among hidden variables that respect the spatial structure of an entire image, and pooling operations to aggregate these over areas of an image. Chaining these operations together in a multilayer hierarchy, we train stacked CRBMs that are able to extract large scale features tuned for a particular object class.

The main contribution of this work is the development of the CRBM model. We modify the standard RBM and learning algorithm to include spatial locality and weight sharing. We develop these in a generative framework for layerwise training of LFDHs. The framework learns a small set of stochastic features that model a distribution over images of a specific object class, which are then used in a discriminative classifier. We demonstrate experimentally that this generative feature learning is effective for discrimination, using the learned features to obtain state-of-the-art performance on object detection tasks. Some parts of this thesis appeared as a conference publication in [26].

Chapter 2

Previous work

The idea of building a hierarchical structure of features for object detection has deep roots in the computer vision literature, with the development of many such models inspired by allusions to the human visual system [16, 9, 21, 34, 25, 30]. We refer to these models generally as local feature detector hierarchies (LFDH). Here we describe Neocognitron, Convolutional Neural Networks, and so-called Biologically inspired models as some specific examples of LFDHs in detail. We discuss how these are different from our stacked CRBMs.

Moreover, some recent related work is presented in this chapter. Greedy layerwise learning, a method for initializing weights of fully connected neural networks [14] and Deep Belief Nets [13] will be described in Section 2.4. The feature learning algorithm that we develop is highly influenced by the aforementioned pioneering works of Hinton et al., and we adopt this layerwise mechanism for LFDH structures. However, before us, Ranzato et al. [30] have developed a different model and employed the layerwise learning mechanism for convolutional neural networks. They used a non-probabilistic encoder/decoder structure called Predictive Sparse Decomposition (PSD) to train each of the layers. Instead, we propose a probabilistic model called Convolutional Restricted Boltzmann Machine (CRBM) for layerwise training. Another related model is Fields of Experts (FoE), a Markov Random Field (MRF) model that resembles a layer of CRBM, and proposed for learning image priors. We discuss how the FoE and CRBM models relate and differ in section 2.6.

An enormous number of methods has been applied to the MNIST handwritten digit dataset. This is also the case for the INRIA human benchmark. Since results of many algorithms on these two popular datasets are available, we are able to compare our results with the performance of different approaches. We will briefly describe some related work for each of these datasets in separate sections.

To our surprise, at the time that our work was under review for a conference[1], we realized that two other independent groups were developing very similar models. Besides our work [26], the CRBM model was developed by both Desjardins and Bengio [8] and Lee et al. [23] roughly at the same time. In the technical report of Desjardins and Bengio, elements such as max pooling, sparsity, and stacking multiple CRBMs in a hierarchy were not included and very limited set of experiments were conducted. However, Lee et al. exploited sparsity, pooling, and hierarchical structure as well. An important component of their model is a specific type of bi-directional pooling layer called *probabilistic max-pooling*. This component enables them to build a complete undirected graphical model called *Convolutional Deep Belief Network (CDBN)*. This model is slightly different from our stacked CRBMs in the way it enforces sparsity and performs pooling.

2.1 Neocognitron

In the late 70s Fukushima designed a neural network architecture, specialized for vision applications, called Neocognitron [9]. His work was influenced mainly by the early work of Hubel and Weisel [16] on the analysis of the cat's visual cortex. Fukushima operationalized the idea of LFDH as a computational model and proposed an unsupervised learning algorithm for this structure.

Fig. 2.1 elaborates the structure of Neocognitron. This is a three stage local feature detector hierarchy, where each stage consists of two layers: filtering/non-linearity and pooling. In Fig. 2.1 the filtering layers are visualized as the links between ($U_0 \rightarrow U_{S1}$), ($U_{C1} \rightarrow U_{S2}$), and ($U_{C2} \rightarrow U_{S3}$), and the pooling layers are the subsequent connections of ($U_{S1} \rightarrow U_{C1}$), ($U_{S2} \rightarrow U_{C2}$), and ($U_{S3} \rightarrow U_{C3}$). Motivated by biology, units that perform local feature computation were called *simple cells* (denoted by U_S) and pooling units were referred to as *complex cells* (denoted by U_C).

As can be seen in Fig. 2.1, simple and complex cells are divided into a number of partitions, each called a *cell plane* in terms of Neocognitron, or *feature map* in our words. The grouping of features is a necessary structural element in all of the LFDH's. Since features are local, each unit can only extract attributes of one local neighborhood of the

[1] Accepted and published as [26].

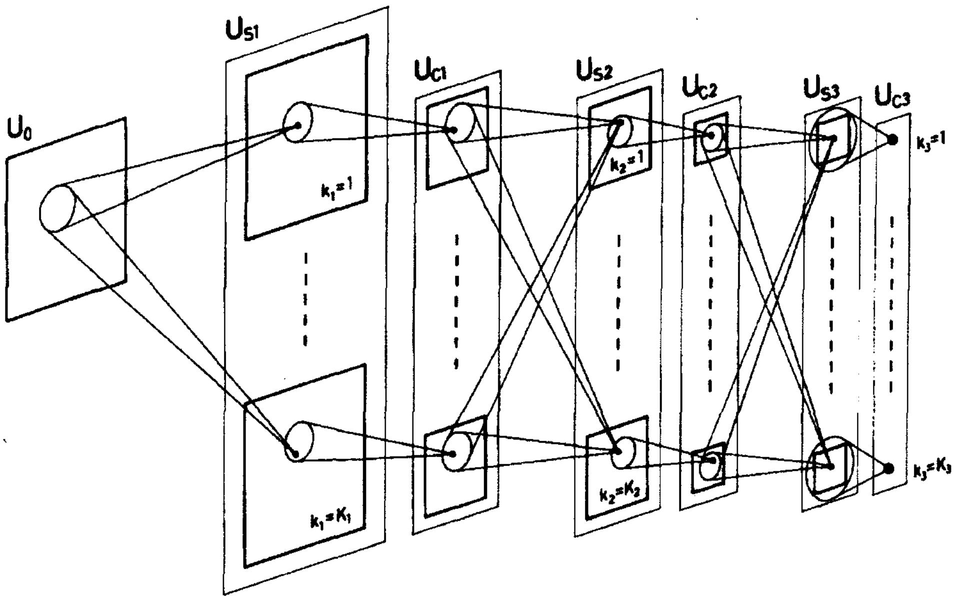


Figure 2.1: Neocognitron, one of the early computational models for hierarchical feature extraction. Figure taken from [9]

input. Therefore parameters of features are shared among a grid of units, so that they can extract the same feature from different locations. We refer to this strategy of sharing the parameters among subsets of units as *weight sharing*. Fukushima describes weight sharing as follows:

> It is assumed that all the cells in a single cell-plane have input synapses of the same spatial distribution, and only the positions of the presynaptic cells are shifted in parallel from cell to cell. Hence, all the cells in a single cell-plane have receptive fields of the same function, but at different positions.[2]

Essentially weight sharing is a way of implementing the filtering operation within locally connected neural networks.

Neocognitron was developed in a decade that computational models were not mature enough. In this model a rather ad-hoc unsupervised learning algorithm is developed to learn weights of simple cells, without any exact criterion for learning e.g., minimizing a defined

[2] One can perceive the strong biological inspiration and commitment of Fukushima in his words.

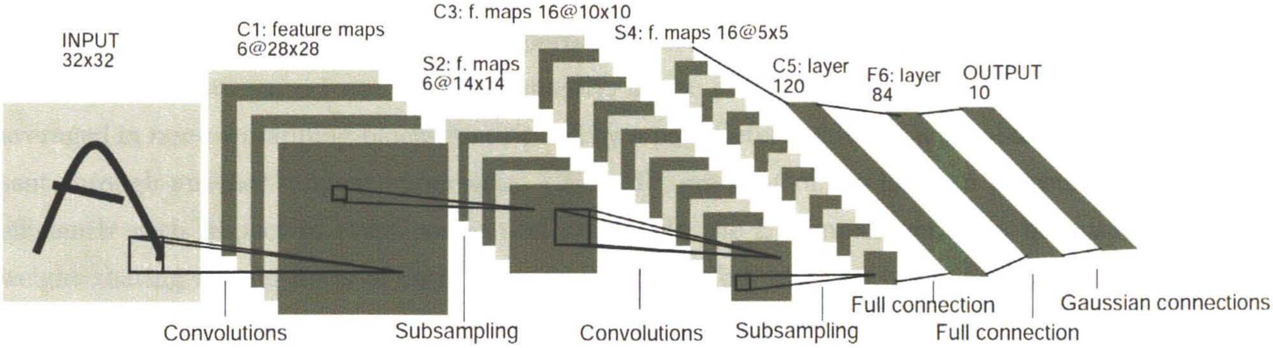


Figure 2.2: Lenet-5, a specialized Neural Network with local connections, weight sharing, and subsampling (pooling) layers for handwritten digit classification. Feature extraction and classification are bundled together in this model. Figure taken from [21]

energy or maximizing the likelihood. Also the model is a bit complicated as it was designed to fit the biological understandings. Inhibitory and Excitatory units were defined, and the activation functions implemented by the simple and complex cells are hard to interpret. However, this model was an opening for more computational investigations.

2.2 Convolutional Neural Networks

LeCun et al. [21] developed the convolutional neural network (CNN), reminiscent of Neocognitron, in which weight sharing is employed with the result that the learned weights play the role of filtering (or convolution) kernels and can be interpreted as a set of adaptable features. This structure is again an LFDH with alternating layers of filtering/non-linearity (convolutional layers) and pooling (subsampling). An important idea of the CNN that distinguishes it from Neocognitron, some later models, and our stacked CRBMs is that the feature extractor and classifier were unified in a single structure. The parameters of both classifier and feature detector were trained globally and supervised by backpropagation (BP). In CNN, after the last stage of feature detection a few fully connected layers were added to perform classification. Hence, the gradient of classification error can be propagated throughout the hierarchy to tune the features.

In figure 2.2 a specific CNN called Lenet-5 is depicted. In Lenet-5, the last few layers with full and Gaussian connections serve as the classifier. This model was proposed for handwritten digit recognition, and achieved a very low error rate on MNIST dataset. In Lenet-5, after each filtering, a tangent hyperbolic function provides non-linearity and maps the responses between -1 and 1. Subsequently, in the subsampling layers, the inputs are averaged in non-overlapping blocks, multiplied by a parameter, added to a tunable bias, and sent through another tangent hyperbolic. Using BP, gradient of the loss can be computed efficiently with respect to every connection weight. In the BP implementation, because of weight sharing among units of each feature map, the effect of those units on the gradient should be summed up.

Globally training a bundled feature extractor and classifier is a neat idea, since the classifier can give useful feedbacks to the feature detectors. However, using BP for gradient based learning of a deep and large CNNs requires many tricks and tweaks. More importantly, it demands lots of labeled data for training. In digit classification, the size of input is pretty small, the setting is really clean (no background clutter, illumination change, etc.), and the number of labeled examples is huge (60000 for MNIST). However, for most realistic vision applications this is not the case. For instance, Ranzato et al. [30] trained a large CNN for object detection (Caltech 101 dataset) using BP starting from random initialization. However, their network obtained a rather poor result, although it could achieve perfect classification performance on the training set. Note that Yann LeCun a founder of Lenet-5 and CNN was also involved in the aforementioned work, so we can be confident that the necessary tweaks for BP were employed.

Shortcoming of CNN. The weak generalization power of CNN when the number of training data is small and the number of free parameters is large, is a case of overfitting or over-parametrization (in words of [30]). Some of the recent work including [30] and ours address this issue by unsupervised layer by layer training of CNNs. It is interesting that in the early 80s, when for example Fukushima's model was around and BP had not become popular, LFDHs were trained unsupervised. Now after decades, another trend of unsupervised learning has been started by the work of Hinton et al. [13, 14] on layerwise learning. However, BP can still be used to fine tune a model that is initialized in unsupervised manner.

2.3 Biologically Inspired Models

HMAX model and its extensions [34, 25] also fall into the category of LFDHs, and are generally referred to as biologically inspired models. HMAX, rather than attempting to learn its bottom-level features, uses hardwired filters designed to emulate visual cortex simple cells. Gabor filters with different orientations and scales build the first filter bank. In the pooling layers the responses are computed using hard Max functions. Max is particularly interesting because it provides invariance while it maintains specificity. In HMAX, the second feature detection layer, compares the signal that reaches it against a set of stored prototypes randomly sampled from images during training.

HMAX is different from CNN and stacked CRBMs developed in this thesis because it does not perform learning on the feature extraction layers. So, it cannot adapt to different problem settings.

2.4 Greedy Layerwise Learning

Hinton et al. proposed a greedy layerwise procedure for training multilayer fully connected neural networks [14] and deep belief nets (DBN) [13]. In this procedure, layers are trained bottom-up and separately, such that each layer is taken out of the network and is viewed as a model called a Restricted Boltzmann Machine (RBM). The RBM has a layer of visible and a layer of hidden variables with full visible-hidden connections. Connection weights of this model are tuned in unsupervised manner and with the criterion of maximum likelihood. The actual input of the network maps into the visible layer of the bottom-most RBM, so we can start by training this layer in unsupervised fashion. Next, the parameters of this first RBM are frozen, and activation probabilities of hidden units given each input example, are inferred. In other words, we induce the first RBM's features (represented by hidden units) for all training data. These features provide observed data for the second RBM, so we can proceed by learning the second RBM. The same procedure can be repeated for the upper layers too. It is worth mentioning that Bengio et al. also developed another variant of greedy layerwise learning based on bottom up training of two layer autoencoders instead of RBMs [3].

Neural Network. Hinton and Salakhutdinov [14] trained the weights of a deep neural network using the above layerwise procedure, and used it as initialization for supervised BP. They reported results 0.2% better than RBF and polynomial kernel SVMs on MNIST, which is assumed significant. It is also over 0.4% better than the best neural network trained using BP on random initialization. Layerwise initialization and subsequent BP were also employed for training deep autoencoders with the goal of dimensionality reduction. This

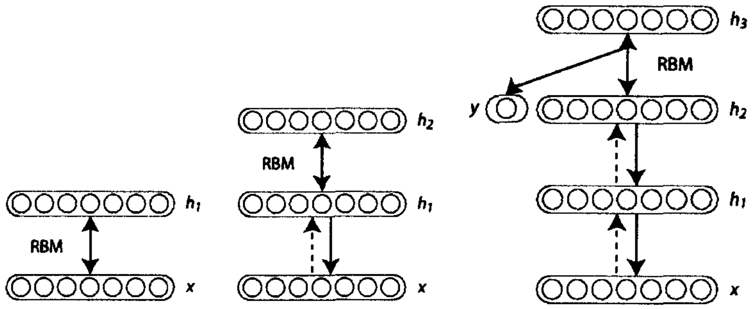


Figure 2.3: An illustration of greedy layerwise learning for DBNs. After learning first and second RBMs for inducing learning h_1 and h_2 features, a third RBM is trained on the h_2 responses and the labels y. [3]

technique was applied to face patches and digit images, and high compression rates achieved.

Deep Belief Net. Another interesting direction of research is to construct multilayer generative models that can actually generate random digits or faces. These models might also do classification by modeling the joint density of data and labels and performing a bayesian decision based on that. Deep Belief Nets (DBN) are multilayer probabilistic generative models with a layer of visible and multiple layers of latent random variables. The top two layers of DBN have undirected connections between them and form an associative memory. The lower layers receive top-down, directed connections from the layer above. The states of the units in the lowest layer represent a data vector [11].

Greedy layerwise learning is also applicable to DBNs. Figure 2.3 illustrates this learning method. For DBNs layerwise learning is theoretically motivated, and under certain conditions stacking RBMs on top of each other improves a variational lower bound on the data likelihood, though the likelihood itself might fall [13]. DBNs were applied to modeling the joint density of handwritten digits and their labels obtaining promising classification results. Also, samples given by this generative model resembled real handwritten digits.

2.5 Predictive Sparse Decomposition

Predictive Sparse Decomposition (PSD) [31] refers to a two layer energy-based non-probabilistic model that consists of a layer of hidden and a layer of visible random variables, with the same functionality and architecture as a continuous RBM, but with a different learning criterion. The PSD is an extension of the famous work of Olshausen & Field [27] on sparse coding of image patches by learning a basis. A problem of the sparse coding model is its inefficient inference. The PSD algorithm aims at learning a basis with low reconstruction error while guaranteeing the efficiency of encoding and sparsity of codes.

Here we describe the PSD model in detail. We denote a state of visible random variables as a vector $\mathbf{x}$ and latent variables as $\mathbf{z}$. A PSD consists of a non-linear encoder representing a directed link from $\mathbf{x}$ to $\mathbf{z}$ and a linear decoder that links $\mathbf{z}$ to $\mathbf{x}$. The encoder maps an input $\mathbf{x}$ into a vector $\mathrm{enc}(\mathbf{x}) = d\tanh(\mathbf{W_e}^\mathsf{T}\mathbf{x})$ where a matrix $\mathbf{W_e}$ and a scalar d are the encoder's parameters, and the Decoder has a parameter matrix $\mathbf{W_d}$ that decodes $\mathbf{z}$ into $\mathrm{dec}(\mathbf{z}) = \mathbf{W_d}^\mathsf{T}\mathbf{z}$. Based on this structure for each configuration of visible and latent variables an energy function is defined as follows:

$$E(\mathbf{x},\mathbf{z};\theta) = \left\|\mathbf{z} - d\tanh(\mathbf{W_e}^\mathsf{T}\mathbf{x})\right\|_2^2 + \alpha\left\|\mathbf{x} - \mathbf{W_d}^\mathsf{T}\mathbf{z}\right\|_2^2 + \lambda\left\|\mathbf{x}\right\|_1. \qquad (2.1)$$

The parameters $\theta = \{\mathbf{W_e}, d, \mathbf{W_d}, \alpha, \lambda\}$, and the above energy for a pair of $\mathbf{x}$ and $\mathbf{z}$ gets a low value (is preferred) if $\mathbf{z}$ is close to the output of encoder $\mathrm{enc}(\mathbf{x})$, $\mathbf{x}$ is close to the output of the decoder $\mathrm{dec}(\mathbf{z})$, and $\mathbf{z}$ is sparse. The first term in the energy accounts for encoding error, the second term for decoding error, and the last term ($L1$ norm) ensures that the codes are sparse. The learning algorithm of PSD is defined to minimize the following loss function:

$$L(\mathbf{x};\theta) = \min_{\mathbf{z}} E(\mathbf{x},\mathbf{z};\theta). \qquad (2.2)$$

To minimize this loss, using an iterative algorithm similar in spirit to EM, we alternate between inferring optimal latent variables according to

$$\mathbf{z}^* = \arg\min_{\mathbf{z}} E(\mathbf{x},\mathbf{z};\theta), \qquad (2.3)$$

and updating the parameters by gradient descent in $E(\mathbf{x},\mathbf{z}^*)$. After the parameters were learned, efficient approximate inference would be done by computing $\mathrm{enc}(\mathbf{x})$. It will give us a good approximation of $\mathbf{z}^*$ because in the energy function (2.1) we made sure that the encoder provides outputs that are close to the optimal codes.

The important difference between PSD and RBM is that PSD is not probabilistic and its loss is defined over the best latent variables, while RBM is probabilistic and it sums over different latent configurations to infer the probability of a data vector. Both classes of probabilistic and non-probabilistic approaches have been used widely in unsupervised learning, and their applicability depends on the application. However, an advantage for the probabilistic approach is that after learning we can actually sample from $\mathbf{z}$ and generate data $\mathbf{x}$ by Markov Chain Monte Carlo (MCMC) methods.

2.5.1 PSD for Learning CNN

Ranzato et al. [30] applied PSD for layerwise learning of CNNs. In their proposed method a grid of overlapping image patches that share at a pooling unit were cropped from training images, and using the above learning procedure, a set of filters were trained that can encode and decode all the patches together and properly. Then they froze the filters and trained another layer of filters on top of the pooling layer. Without any further fine-tuning they achieved good results on generic object detection and handwritten classification. They also fine-tuned the weights using BP and showed that fine-tuning improves the results.

Our proposed CRBM, in contrast to this line of work, defines a probabilistic model over images and features using the criterion of modeling the distribution over input images rather than a non-probabilistic loss minimization. Further, explicit shift-invariance is built into the PSD-based work, whereas in our model, shift-invariance is implicitly handled by learning filters from full images instead of patches.

2.6 Fields of Experts

Roth and Black [32] developed the *Fields of Experts* (FoE) model, for use in constructing image priors, with applications to image denoising and image inpaining. This model defines a probability distribution over entire images as a product of patch potentials. Consider a FoE with N fixed-size filters given by $\{\mathbf{J_i}\}_{i=1}^{N}$. We denote a patch of an image $\mathbf{X}$ which is located below and to the right of a pixel k and have the same size as the filters, by $\mathbf{X_{(k)}}$. Because $\mathbf{J_i}$ and $\mathbf{X_{(k)}}$ are equal size, the result of filtering $\mathbf{X_{(k)}}$ by $\mathbf{J_i}$ is just a scalar denoted by $\mathbf{J_i} \odot \mathbf{X_{(k)}}$. Each filter $\mathbf{J_i}$ impose a potential function on the patch $\mathbf{X_{(k)}}$ given by

$$\phi\left(\mathbf{X_{(k)}}|\mathbf{J_i}, \alpha_i\right) = \left(1 + \frac{1}{2}\left[\mathbf{J_i} \odot \mathbf{X_{(k)}}\right]^2\right)^{-\alpha_i}. \tag{2.4}$$

This potential aims at capturing the harmony of natural images' filter responses $\left[\mathbf{J_i} \odot \mathbf{X_{(k)}}\right]$ by inducing a student-t distribution on them. To measure the probability of an image we

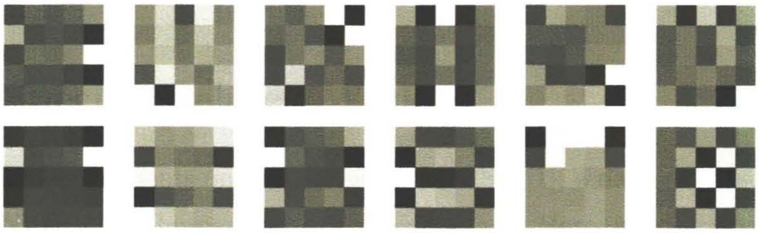


Figure 2.4: Selection of the 5×5 filters obtained by training the Fields-of-Experts model on a generic image database. [32]

iterate over all the patches and filters, and multiply their potentials:

$$p(\mathbf{X};\theta) = \frac{1}{Z(\theta)} \prod_k \prod_{i=1}^{N} \phi(\mathbf{X}_{(\mathbf{k})}|\mathbf{J}_\mathbf{i}, \alpha_i). \tag{2.5}$$

Here $Z(\theta)$ is the normalization constant to make it a probability over all possible $\mathbf{X}$'s, and parameters are $\theta = \{\mathbf{J}_\mathbf{i}, \alpha_i\}_{i=1}^{M}$, set of filters and their corresponding exponents.

Now that the model is defined, parameters can be estimated to maximize the likelihood (2.5) over θ given a set of training images. However because computing the partition function is intractable, approximate methods should be employed. Roth & Black used Contrastive Divergence learning [10] as an approximation of maximum likelihood, which resulted in the filters shown in Fig. 2.4.

FoEs with student-t potentials and CRBMs are similar in spirit, owing to the idea of weight sharing that they both adopt (a more general term might be *expert sharing* for FoE). However, they are different mainly because of their distinct patch potentials. Similarly, RBM and product of student-t experts [40] are different. This difference is crucial because as noted in [39] the quadratic potential function of FoE model favors filters with close to zero responses on training images and non-zero responses on noisy ones. In contrast, the CRBM's patch potential favors filters with high response on training images and low responses elsewhere. This leads to extracting frequent patterns of training images e.g., oriented edges as in Fig. 5.4 in place of noisy filters shown in Fig. 2.4. One might argue that FoE is a general model which is not limited to a particular potential such as student-t. This is in fact true and we can characterize CRBM as an instance of FoE. But, CRBM on the other hand is a two layer model that has a layer of latent variables, playing very important role in this model. Thus we prefer not to describe CRBM as a FoE, which does not explicitly have latent variables and is widely known for constructing image priors.

2.7 Task Specific Related Work

We conducted our experiments on two datasets: MNIST [1]; a dataset of isolated handwritten digits and INRIA [7]; a human detection benchmark. Here, we describe state-of-the-art previous work on the same datasets. In Chapter 5 we will compare our results with these models.

2.7.1 MNIST Handwritten Digits

An enormous number of methods has been applied to the MNIST digit dataset. The aforementioned neural network-based approaches such as [31, 30] attain some of the best results on this dataset. For smaller numbers of training images, the *patchwork of parts* (PoP) model of Amit and Trouvé [2], which learns a deformable model of edge parts, attains excellent accuracy. Our method obtains results competitive with the state-of-the-art on MNIST.

2.7.2 INRIA Pedestrian Benchmark

A substantial volume of previous work on pedestrian detection also exists. The state-of-the-art results on the INRIA pedestrian detection dataset include Dalal & Triggs [7] and Tuzel et al. [37]. Dalal & Triggs extract HOG descriptors from images and train a linear SVM on them in the framework of sliding window approach to discriminate humans from other stuff. Tuzel et al. use a set of covariance descriptors describing the statistics of pixels in sub-regions of a pedestrian image, and develop a classifier for the Riemannian manifold on which such covariance descriptors lie. Our work focuses on automatically learning a set of features for detection, and obtains similar results with a generic SVM-based classifier.

Chapter 3

Preliminaries

3.1 Restricted Boltzmann Machine

The Restricted Boltzmann Machine (RBM) [35] is a two layer undirected graphical model that consists of a layer of observed and a layer of hidden random variables, with a full set of connections between them (depicted in Fig. 3.1a). It is a generative framework that models a distribution over visible variables by introducing a set of binary stochastic hidden units representing presence of features. We can associate the RBM's observed variables with image patches as shown in Fig. 3.1b. In this case the RBM's weights would represent a set of filters.

Typically hidden variables ($\mathbf{h}$) of an RBM are binary, turning a particular feature on or off. In the original RBM visible random variables ($\mathbf{v}$) were also binary. In this thesis we refer to this model as a *binary RBM* or just RBM. A modification of the binary RBM makes it suitable for modeling a density over continuous visible variables, while hidden units are still binary [6]. We call this second model *continuous RBM.* A continuous RBM is appropriate for modeling natural images at pixel level, while a binary RBM can model a density over hidden units of another RBM or quasi-binary images (e.g., handwritten digits).

A key characteristic of the RBM is that its hidden units are conditionally independent given the observed data. This property makes each hidden unit an independent expert on detecting a specific feature. Therefore, the RBM is an instance of the product of experts model [10].

The RBM is a probabilistic energy-based model. The probability of observed variables in an RBM with parameter set θ is defined according to joint energy of visible and hidden

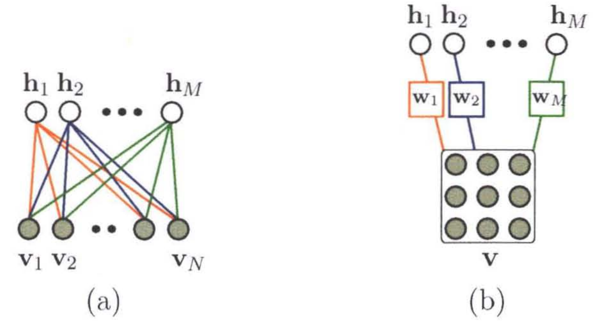

$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_M$
$\mathbf{v}_1$ $\mathbf{v}_2$ $\mathbf{v}_N$
(a)
$\mathbf{w}_1$ $\mathbf{w}_2$ $\mathbf{w}_M$
$\mathbf{v}$
(b)

Figure 3.1: **(a)** An RBM with a layer of observed random variables $\mathbf{v}$, and a layer of binary hidden random variables $\mathbf{h}$. **(b)** Observed variables of an RBM can be associated with image patches with the result that weight connections represent a set of filters.

units $E(\mathbf{v}, \mathbf{h}; \theta)$, as a Gibbs distribution

$$p(\mathbf{v}; \theta) = \frac{1}{Z(\theta)} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)} , \tag{3.1}$$

where $\mathbf{v}$ and $\mathbf{h}$ denote vectors of visible and hidden variables, $Z(\theta)$ is the partition function, and the energy function is defined depending on whether the visible variables are continuous or binary. The energy functions of the binary and continuous RBMs are defined as E_1 and E_2 respectively,

$$E_1(\mathbf{v}, \mathbf{h}; \theta) = -\sum_{i,j} \mathbf{v}_i \mathbf{W}_{ij} \mathbf{h}_j - \sum_i \mathbf{b}_i \mathbf{v}_i - \sum_j \mathbf{c}_j \mathbf{h}_j , \tag{3.2}$$

$$E_2(\mathbf{v}, \mathbf{h}; \theta) = E_1(\mathbf{v}, \mathbf{h}; \theta) + \frac{1}{2} \sum_i \mathbf{v}_i{}^2 , \tag{3.3}$$

where index i iterates over observed variables and j iterate over hidden units, and model parameters are $\theta = \{\mathbf{W}, \mathbf{b}, \mathbf{c}\}$. The matrix $\mathbf{W}$ determines the symmetric interaction between pairs of hidden and visible units, and parameters $\mathbf{b}$ and $\mathbf{c}$ are the bias terms that set the unary potential of the units.

Inference in RBMs is straightforward. In the binary RBM conditionals are derived as

$$p(\mathbf{h}_j = 1 | \mathbf{v}) = \sigma(\mathbf{c}_j + \sum_i \mathbf{v}_i \mathbf{W}_{ij}) , \tag{3.4}$$

$$p_1(\mathbf{v}_i = 1 | \mathbf{h}) = \sigma(\mathbf{b}_i + \sum_j \mathbf{W}_{ij} \mathbf{h}_j) , \tag{3.5}$$

where $\sigma(x) = 1/(1+e^{-x})$ is the logistic sigmoid function. For the continuous RBM, Eq. (3.4) still holds, but the conditional distribution of visible units becomes a normal,

$$p_2(\mathbf{v}_i|\mathbf{h}) = \mathcal{N}(\mathbf{b}_i + \sum_j \mathbf{W}_{ij}\mathbf{h}_j, 1). \tag{3.6}$$

Here the variance is set to one because in a pre-processing stage visible units can be scaled with an arbitrary constant value.

3.2 Contrastive Divergence Learning

Ideally we want to learn RBM parameters by maximizing the likelihood in a gradient ascent procedure. The gradient of the log-likelihood for an energy-based model on data $\mathbf{v}$ is

$$\frac{\partial}{\partial\theta}L(\theta) = -\left\langle\frac{\partial E(\mathbf{v};\theta)}{\partial\theta}\right\rangle_{data} + \left\langle\frac{\partial E(\mathbf{v};\theta)}{\partial\theta}\right\rangle_{model}, \tag{3.7}$$

where $E(\mathbf{v};\theta)$ is the free energy of $\mathbf{v}$, and $\langle.\rangle_{data}$ and $\langle.\rangle_{model}$ denote the expected value over all possible visible vectors $\mathbf{v}$ with respect to the training data and the model distributions. Intuitively the goal of learning is to push the energy on the training data points down and pull the energy elsewhere up. In (3.7) the first term accounts for pushing down the energy on the data points, and the second term pulls the energy up of the vectors that have low energy according to the model but do not have high probability in the training data.

For an RBM the free energy is derived from (3.1) as

$$E(\mathbf{v};\theta) = -\log\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)} \tag{3.8}$$

and the gradient of free energy takes the form

$$\frac{\partial E(\mathbf{v};\theta)}{\partial\theta} = \sum_{\mathbf{h}} p(\mathbf{h}|\mathbf{v};\theta)\frac{\partial E(\mathbf{v},\mathbf{h};\theta)}{\partial\theta}. \tag{3.9}$$

Although the sum over $\mathbf{h}$ is a sum over an exponential number of states of a binary vector, this sum can be computed efficiently because $p(\mathbf{h}|\mathbf{v};\theta)$ decomposes into $\prod_j p(\mathbf{h}_j|\mathbf{v};\theta)$ and $E(\mathbf{v},\mathbf{h};\theta)$ breaks into $\sum_j E_j(\mathbf{v},\mathbf{h}_j;\theta_j)$. Thus the gradient also decomposes into

$$\frac{\partial E(\mathbf{v};\theta)}{\partial\theta_j} = \sum_{\mathbf{h}_j} p(\mathbf{h}_j|\mathbf{v};\theta)\frac{\partial E_j(\mathbf{v},\mathbf{h}_j;\theta)}{\partial\theta_j}. \tag{3.10}$$

Note that $\mathbf{h}_j$ can only take values from zero and one, thus the sum over it is only the sum over two terms. This gives us an efficient way for computing the first term of (3.7), but unfortunately computing the second term, the expected value regarding an RBM distribution, involves iterating over exponentially many visible states, which makes it intractable. However, Hinton [10] proposed another objective function called *contrastive divergence* (CD) that can be efficiently minimized during training as an approximation to maximizing the likelihood.

During contrastive divergence learning a Gibbs sampler is initialized at each data point and is run for one or a few steps (n) to obtain an approximation of the model distribution. This approximation is plugged into second term of (3.7) to obtain

$$\frac{\partial}{\partial\theta}L(\theta) = -\left\langle \frac{\partial E(\mathbf{v};\theta)}{\partial\theta} \right\rangle_{data} + \left\langle \frac{\partial E(\mathbf{v};\theta)}{\partial\theta} \right\rangle_{gibbs(n)} , \tag{3.11}$$

where $\langle . \rangle_{gibbs(n)}$ represents expected value with respect to the samples provided by n steps of Gibbs sampling initiated at training data. Note that CD is not an unbiased estimator of likelihood gradient, but it increases a lower bound of the likelihood [10] and performs well in practice.

For an RBM the CD update rule of the weight $\mathbf{W}_{ij}$ becomes

$$\mathbf{W}_{ij} = \mathbf{W}_{ij} + \eta \left(\langle \mathbf{v}_i^0 \mathbf{h}_j^0 \rangle - \langle \mathbf{v}_i^n \mathbf{h}_j^n \rangle \right) , \tag{3.12}$$

where η is the learning rate, the random variable $\mathbf{v}^0$ takes value from the data distribution, $\mathbf{h}^0$ is obtained from $p(\mathbf{h}^0|\mathbf{v}^0)$, random variable $\mathbf{v}^n$ takes value from sampled data generated by n full steps of Gibbs sampling (basically sampled from $p(\mathbf{v}^n|\mathbf{h}^{n-1})$), and $\mathbf{h}^n$ is obtained according to $p(\mathbf{h}^n|\mathbf{v}^n)$. For training RBMs usually employing only one step of Gibbs sampling performs well, but the number of Gibbs sampling steps can be increased as the learning proceeds. This might require a change of the learning rate as well.

3.3 Layerwise Training of Fully Connected Nets

As described in Section 2.4, RBMs can be stacked on top of each other to pretrain a supervised Neural Network, an autoencoder, or a Deep Belief Net (DBN). Consider a fully connected network with an input layer $\mathbf{v}$ and a number of hidden layers $\mathbf{h_1}, \mathbf{h_2}, \ldots$. We can start by training an RBM $(\mathbf{v}, \mathbf{h}_1; \theta_1)$ between $\mathbf{v}$ and $\mathbf{h}$, which has parameters θ_1. The training data is used for unsupervised training of this bottom-most model. Next, θ_1 if frozen and a second RBM $(\mathbf{h}_1, \mathbf{h}_2; \theta_2)$ is trained with the visible layer $\mathbf{h}_1$. The data for training this second RBM are obtained from $p(\mathbf{h}_1|\mathbf{v}; \theta_1)$. Additional layers can also be added on top of the model in the same way. After this greedy layerwise learning, BP or wake-sleep learning [12] might be employed to fine-tune the parameters.

Chapter 4

Convolutional RBM

In the standard RBM all observed variables are related to all hidden variables by different parameters. Using an RBM to extract global features from full images for object detection is not promissing considering how large the images are. Describing images in terms of spatially local features needs fewer parameters, generalizes better, and offers re-usability as identical local features can be extracted from different locations of an image. One approach is to train RBMs on patches sampled from images to create local features (see Fig. 3.1b). We refer to this approach as pach-based RBM. The patch-based RBM does not respect the spatial relationship of image patches, and considers each image patch independent from the nearby patches. Therefore, features extracted from neighboring patches become independent and potentially redundant.

To tackle this problem, we introduce an extension of the RBM model, called the Convolutional RBM (CRBM). The CRBM, similar to the RBM, is a two layer model in which visible and hidden random variables are structured as matrices. Therefore, in this model locality and neighborhood are definable both for hidden and visible units. The CRBM's visible matrix could represent an image and subwindows of it would denote image patches. The CRBM's hidden-visible connections are local and weights are shared among clusters of the hidden units. Fig. 4.1 illustrates a CRBM. Each cluster of hidden units is called a *feature map*. A feature map is a binary matrix representing presence of a single feature at different locations of the input. Therefore, hidden units are divided into different feature maps to denote presence of different features at different locations of the visible units.

The CRBM's hidden units extract features from overlapping patches of visible units and features of neighboring patches complement each other and cooperate to model the input.

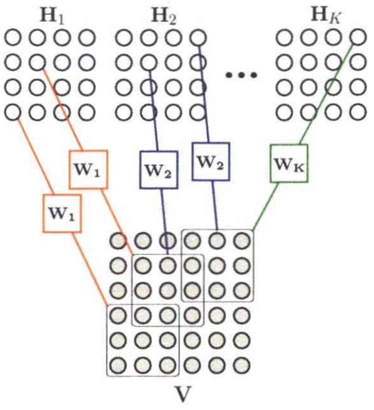


Figure 4.1: A CRBM with a matrix of visible units $\mathbf{V}$ and a matrix of hidden units $\mathbf{H}$ that are connected via K 3×3 filters: $\mathbf{W_1}, \mathbf{W_2}, \dots, \mathbf{W_K}$. The hidden units are partitioned into K submatrices called feature maps: $\mathbf{H_1}, \mathbf{H_2}, \dots, \mathbf{H_K}$. Each hidden unit represents presence of a particular feature at a 3×3 neighborhood of visible units. Units within a feature map represent the same feature at different locations of $\mathbf{V}$.

Unlike a patch-based RBM, a CRBM is trained on complete images or large regions of them to learn local features and exploit spatial relationship of overlapping patches. Because the hidden units of overlapping patches co-operate in a CRBM, an image pattern that is explained by one hidden unit at one neighborhood does not demand to be explained again in another overlapping patch. This reduces redundancy of features. Compare this with the case of a patch-based RBM that tries to model each image patch independently. The connection scheme that we employed in a CRBM is known as convolutional connections, and has been used in models such as [21], but a significant difference here is that the convolutional connection is employed in a generative partially observed MRF architecture.

In a CRBM, hidden units denoted by matrix $\mathbf{H}$, are binary and represent presence/absence of local features in subwindows of visible units $\mathbf{V}$. We divide the hidden units into submatrices each called a feature map. Assume $\mathbf{H}$ consists of K feature maps $\{\mathbf{H_k}\}_{k=1}^{K}$. The variables of each feature map are connected via identical $x \times y$ filters to different $x \times y$ neighborhoods of observed variables e.g. pixels (see Fig. 4.1). Hence, binary hidden units of each feature map $\mathbf{H_k}$ represent presence of a single local feature at different locations of $\mathbf{V}$. We denote parameters of the k^{th} filter that links $\mathbf{H_k}$ units to different neighborhoods of image $\mathbf{V}$ as $\mathbf{W_k}$.

Notation. Note that because of the intuitive application of CRBM for modeling images we sometimes use the word "image" to refer to the visible matrix $\mathbf{V}$ and word "pixel" to mention a single visible variable. To formulate the energy function of a CRBM, we need to point at a certain subwindow of visible matrix $\mathbf{V}$. Let $\mathbf{V_{(q)}}$ be an $x \times y$ subwindow of image $\mathbf{V}$ with top-left corner at pixel q. Later we will use the notation $\mathbf{H_{k(r)}}$ to denote a $x \times y$ subwindow of the k^{th} feature map with top-left corner at node r. Let $\mathbf{A} \odot \mathbf{B}$ be the generalization of dot product, defined only for the matrices of the same size, as the dot product of their vectorized forms. Thus $\mathbf{A} \odot \mathbf{B} = \mathbf{a}^{\mathsf{T}}\mathbf{b}$ where $\mathbf{a}$ and $\mathbf{b}$ are the vectors obtained by concatenating elements of $\mathbf{A}$ and $\mathbf{B}$. We define another operator $\mathit{Filter}(\mathbf{C}, \mathbf{D})$ that filters matrix $\mathbf{C}$ with filtering kernel $\mathbf{D}$. This filtering is valid i.e., performed without extending the input matrix $\mathbf{C}$, so the result will be smaller than $\mathbf{C}$.

4.1 CRBM as a Probabilistic Model

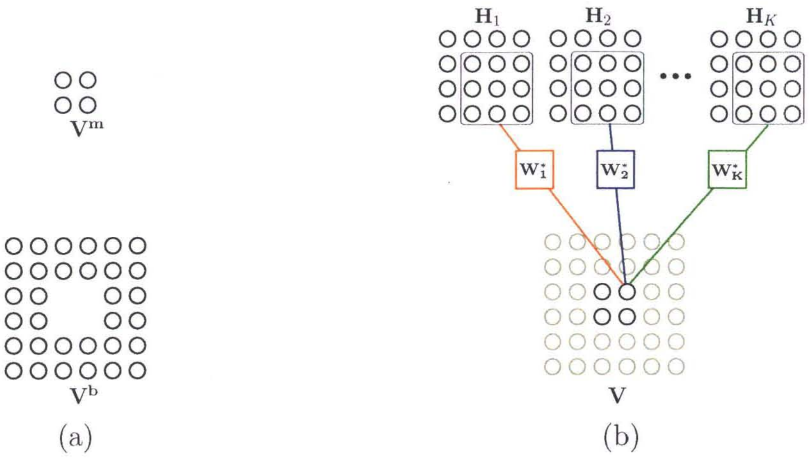
The CRBM is a probabilistic energy based model that induces a density over matrices according to the joint energy of visible variables and hidden variables. This joint energy is defined as the negative sum of pairwise interactions between pairs of hidden and visible nodes, added to unary potentials. The pairwise interactions are determined based on a set of filters that describe the scheme and strength of hidden-visible local links. We can write the joint energy of the CRBM as

$$E_3(\mathbf{V}, \mathbf{H}; \theta) = -\sum_{k,q} \mathbf{H}_{kq} \left(\mathbf{W_k} \odot \mathbf{V_{(q)}}\right) -\sum_{i} b\mathbf{V}_i -\sum_{k,q} \mathbf{c}_k \mathbf{H}_{kq}. \tag{4.1}$$

Here index q iterates over pixels of $\mathbf{V}$ with valid $\mathbf{V_{(q)}}$, it also iterates over nodes of $\mathbf{H_k}$, $\theta = \{\{\mathbf{W_k}\}, \mathbf{c}, b\}$, and i iterates over all the pixels of $\mathbf{V}$. Essentially, a continuous variant of CRBM can be defined with an energy modification similar to (3.3). In the energy function of (4.1), identical bias term is applied to all the visible units and biases are shared among the hidden units in each feature map. This is due to our assumption that different visible variables have similar statistics and features are equally probable at different locations of images. When this is not the case, specific bias terms can be associated with different units in the energy function.

Given (4.1) the conditional probability of a hidden unit given $\mathbf{V}$ takes the form of non-linear filtering

$$p(\mathbf{H}_{kq}\!=\!1|\mathbf{V}) = \sigma\left(\mathbf{W_k} \odot \mathbf{V_{(q)}} + \mathbf{c}_k\right). \tag{4.2}$$



(a) (b)

Figure 4.2: **(a)** Observable units are divided into middle $\mathbf{V^m}$ and boundary $\mathbf{V^b}$ regions. **(b)** A CRBM with 3×3 filters from the view of visible units. Visible units are connected to 3×3 neighborhoods of hidden units by horizontally and vertically flipped versions of original filters denoted by $\{\mathbf{W^*_k}\}$. We can sample from units of $\mathbf{V^m}$ having the configuration of hidden unit using the flipped filters.

The conditional probability of visible units needs more careful treatment, because the boundary units are within a smaller number of subwindows compared to the interior pixels. As an extreme case, the top left pixel only appears in K patches, while a middle pixel may contribute to Kxy features. This problem is caused by the asymmetry of the CRBM connections from the view of visible units. To make the model symmetric, one may extend the CRBM's connections to an infinite MRF, but this approach requires unbounded training images! Instead, we divide the visible units into two partitions: boundary and middle variables (Fig. 4.2a). Let $\mathbf{V^b}$ denote a strip of boundary variables including $x-1$ margin pixels from left and right, and $y-1$ pixels form top and bottom of $\mathbf{V}$, ant $\mathbf{V^m}$ represents the other interior pixels. Clearly all nodes in $\mathbf{V^m}$ are connected with the same set of filters to different feature maps as depicted in Fig. 4.2b. These filters, given by $\{\mathbf{W}^*_k\}$, are horizontally and vertically flipped versions of the original filters.

For binary CRBMs the conditional probability of a middle observable unit at position r is

$$p(\mathbf{V^m}_r = 1|\mathbf{H}) = \sigma\left(\sum_k \mathbf{W^*_K} \odot \mathbf{H_{k(r)}} + b\right). \tag{4.3}$$

However, the conditional probabilities of the boundary pixels cannot be computed accurately as noted above. Later we describe how computation of these probabilities can be avoided.

4.2 Learning CRBM Parameters

An important characteristic of filters learned using a CRBM is that they are shift invariant, meaning that none of the filters can be reconstructed by translating another filter. Shift invariance is a desired property of visual features because it can considerably decrease the number of features needed. In a CRBM this property arises from the fact that each filter is applied to all overlapping neighborhoods of an image. Hence, learning two filters that are translations of each other, is unlikely since it does not increase the likelihood of the filters.

The CRBM parameters $\{\mathbf{W_k}\}$, $\mathbf{c}$, and b are learned by minimizing the contrastive divergence (CD) over a set of training images. We do the required Gibbs sampling from the CRBM distribution by sampling from hidden variables given the visible ones, and next from observed variables given the hidden ones. However, we do not have enough features to describe the conditional distribution of boundary visible units, so we cannot sample from them precisely. Further, doing a number of Gibbs sampling steps might cause uninformative samples of the boundary pixels to propagate to other pixels.

Therefore, instead of maximizing the full data log-likelihood we (approximately) maximize the log-likelihood conditional on the image boundaries $\sum_{\mathbf{V}} \log p(\mathbf{V^m}|\mathbf{V^b};\theta)$. Under the new setup, CD learning is still possible since we can sample from the conditional distribution $p(\mathbf{V^m}|\mathbf{V^b};\theta)$ by n full Gibbs sampling steps. This sampling is done by alternating between sampling $\mathbf{H}$ given $\mathbf{V}$ (4.2) and sampling interior image region $\mathbf{V^m}$ given $\mathbf{H}$ (4.3). Then the image boundary is concatenated with the interior pixels to provide data for other sampling steps.

4.2.1 Computing Gradients

To perform CD learning for CRBM we need to compute $\partial E_3(\mathbf{V};\theta)/\partial\theta$ and plug it into Eq. 3.11. Here we only write down the gradients for the weight matrix $\mathbf{W}$. Other parameters can be learned in a similar way. The gradient of energy with respect to the k^{th} filter $\mathbf{W_k}$ is

$$\frac{\partial E_3(\mathbf{V};\theta)}{\partial \mathbf{W_k}} = \sum_{\mathbf{H_k}} p(\mathbf{H_k}|\mathbf{V}) \frac{\partial E_{3k}(\mathbf{V},\mathbf{H_k};\theta)}{\partial \mathbf{W_k}} \tag{4.4}$$

where E_{3k} is the part of E_3 that involves $\mathbf{W_k}$ and $\mathbf{H_k}$ units. Because of the conditional independency of hidden variables the energy can be decomposed into the desired components.

Conveniently, the gradient of the joint energy can be written as a filtering operation:

$$\frac{\partial E_{3k}(\mathbf{V}, \mathbf{H_k}; \theta)}{\partial \mathbf{W_k}} = -\mathit{Filter}(\mathbf{V}, \mathbf{H_k}) \tag{4.5}$$

Further, $p(\mathbf{H_k}|\mathbf{V})$ decomposes into $\prod_q p(\mathbf{H}_{kq}|\mathbf{V})$. Let $\mathbf{P}(\mathbf{H_k} = 1|\mathbf{V})$ denote a matrix in which conditional probabilities of individual hidden units are placed at the same position as they are located in $\mathbf{H_k}$. Having defined this, we would rewrite (4.4) as

$$\frac{\partial E_3(\mathbf{V}; \theta)}{\partial \mathbf{W_k}} = -\mathit{Filter}\left(\mathbf{V}, \mathbf{P}\left(\mathbf{H_k} = 1|\mathbf{V}\right)\right) \tag{4.6}$$

Substituting (4.6) into the formula of CD gradient (3.11) gives us an elegant update rule for weights of a CRBM. All the primitives for CD learning can be expressed as filtering operations. To compute matrices $\mathbf{P}(\mathbf{H_k} = 1|\mathbf{V})$ we need to filter $\mathbf{V}$ and $\mathbf{W}$, and to obtain $\mathbf{P}(\mathbf{V^m} = 1|\mathbf{H})$ we filter $\mathbf{H}$ and $\mathbf{W}^*$. The gradient of energy also took the form of filtering (4.6). Because of this, learning of CRBMs is relatively fast, because filtering can be computed efficiently using graphics processing unit (GPU) or available utilized codes.

4.2.2 Pseudocode

The pseudocode of learning matrix $\mathbf{W}$ for binary CRBMs, using one step of Gibbs sampling, is provided in Alg. 1:

Algorithm 1 Stochastic CD gradient descent for learning binary CRBM's weights.

```
Input: Learning rate $\eta$, data matrix $\mathbf{V0}$, and filters $\{\mathbf{W_k}\}_{k=1}^{K}$.
Output: New estimate of $\{\mathbf{W_k}\}_{k=1}^{K}$.
for $k = 1$ to $K$ do
  $\mathbf{PH0_k} \leftarrow \sigma\left[\mathit{Filter}(\mathbf{V0}, \mathbf{W_k}) + \mathbf{c}_k\right]$
  $\mathbf{Grad0_k} \leftarrow \mathit{Filter}(\mathbf{V0}, \mathbf{PH0_k})$
  $\mathbf{H0_k} \sim \mathit{Bernoulli}(\mathbf{PH0_k})$
end for
$\mathbf{V1}^m \leftarrow \sigma\left[\sum_{k=1}^{K} \mathit{Filter}(\mathbf{H0_k}, \mathbf{W_k^*}) + b\right]$
$\mathbf{V1} \leftarrow \mathit{Concatenate}(\mathbf{V0^b}, \mathbf{V1}^m)$
for $k = 1$ to $K$ do
  $\mathbf{PH1_k} \leftarrow \sigma\left[\mathit{Filter}(\mathbf{V1}, \mathbf{W_k}) + \mathbf{c}_k\right]$
  $\mathbf{Grad1_k} \leftarrow \mathit{Filter}(\mathbf{V1}, \mathbf{PH1_k})$
  $\mathbf{W_k} \leftarrow \mathbf{W_k} + \eta(\mathbf{Grad0_k} - \mathbf{Grad1_k})$
end for
```

4.3 Sparsity

An issue in learning CRBMs is the overcompleteness of its feature representation. Note that because of the convolutional connections, the feature space of a CRBM with K feature maps is almost K times overcomplete. For the case of continuous CRBM this problem is less severe since modeling real values using binary hidden variables does not lead to overcompleteness. The CD learning can be used for learning overcomplete representations [36], but our experiments showed that it cannot handle the highly overcomplete representation of the CRBMs. From the perspective of learning features for later classification, we would like to extract as many features as we can. However when employing many features, after a few iterations of CD parameter updates, sampled images become almost identical to the original ones, and the learning signal weakens significantly. Increasing the number of Gibbs sampling steps in CD learning is a solution to this problem; but it is time consuming and increases the variance of the gradient estimate. As an alternative solution, we enforced sparsity on the hidden unit activities, which decreases the information content of the feature representation. This helps the learning signal to become stronger and more hidden units to contribute in reconstruction of the images.

For learning an overcomplete basis for natural images Olshausen & Field [27] proposed to encouredge the coeficients in the linear combination to be sparse. Motivated by the idea of sparse coding, Lee et al. [22] developed a sparse variant of RBMs. In their model the $\mathbf{c}$ parameters that control the sparsity of hidden units, were tuned at each learning iteration to obtain a fixed small activation probability for hidden units. Also in [19], sparsity is added to RBMs by continuously decreasing a small constant from hidden bias terms. In our experiments, we employed two tricks: First, freezing the $\mathbf{c}$ parameters at a negative fixed constant and second, constraining $\mathbf{c}$ to be smaller than a fixed upper bound but letting it be learned during training. These two modifications performed similarly and significantly improved the features learned in a CRBM.

4.4 Stacked CRBMs

Our hierarchy of CRBMs is trained layerwise and bottom-up in a procedure similar to the layerwise learning of fully connected networks (see Sections 2.4 and 3.3). One difference here is that we use CRBM as the building block of our local feature detector hierarchy whereas in fully connected networks RBM is used. Further, after each CRBM filtering layer, deterministic subsampling is implemented. To subsample the features we perform Max pooling in non-overlapping image regions. The pooling layer causes the feature dimensionality to decrease, as opposed to filtering that might increase it. Moreover, it makes the features robust to small shifts and distortions and lets the higher level features grow over regions of the input image.

Fig. 4.3 schematically illustrates the greedy layerwise learning for stacked CRBMs and a framework in which their features can be used for classification. Learning flows left to right in Fig. 4.3 and basically from lower levels upward. First a layer of CRBM with parameters θ_1 is trained on the training images and a set of representative features is learned. Then parameters of this CRBM are frozen and the probability of hidden units being activated are computed as a matrix $\mathbf{P}(\mathbf{H} = 1|\mathbf{V}; \theta_1)$. Values in this matrix are all between 0 and 1 and are obtained by applying a logistic sigmoid non-linearity on top of filtering. Next, the probability matrix is downsampled by taking the maximum values in local windows. Another layer of CRBM, in the same fashion, is trained on top of the subsampled probabilities, followed by another max pooling layer. We stop after the fourth layer, and use the final responses as the input of a discriminative classifier.

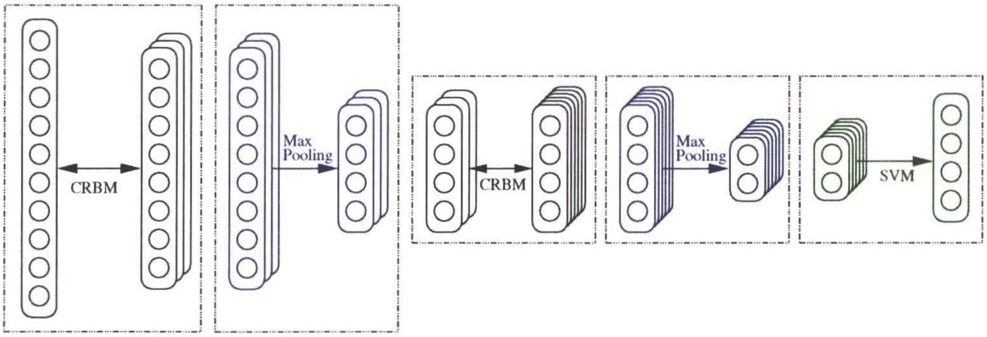


Figure 4.3: A schematic illustration of greedy layerwise learning for stacked CRBMs. Learning proceeds from left to right. Note that Max pooling layers are deterministic and do not have any tunable parameter. The first and third layers are the CRBM models learned by minimizing CD. A SVM is trained using labeled data on the outputs of the feature detector hierarchy.

Chapter 5

Details and Experiments

We experimentally evaluated the discriminative strength of the proposed hierarchical feature learning method. In this chapter we present the experimental results and some details of training the stacked CRBMs. We performed two sets of experiments on handwritten digit recognition (MNIST dataset) and pedestrian detection (INRIA human dataset). Obtaining state-of-the-art accuracy in these two different tasks demonstrates the robustness of our feature learning algorithm and its capability to extract task-specific features.

Model. For each task, we construct a separate four layer feature extractor in the bottom-up layerwise manner. The second and fourth layers of this hierarchy are deterministic max pooling layers that do not have any tunable parameter. We fix the subsampling window size ad-hoc and based on the input size. The first and third layers are the filtering connections that we train as the CRBM models. The weights of these layers were tuned by CD learning with one step of Gibbs sampling. Finally, we trained a discriminative classifier (SVM) on the last layer's outputs to do classification. For handwritten digits, we employed RBF kernel SVM in the one-vs-rest fashion. For pedestrian detection, we combined our features with fine-scale HOG descriptors [7] and trained a linear SVM on the combination of these two features.

Relaxing sparsity. Ranzato et al. [30] reported that although generative feature learning procedure benefits from a sparsifying non-linearity, the final discriminative classifier achieves better accuracy when the non-linearity is relaxed and features become less sparse. Our experiments also supported this relaxation. Thus, after the feature learning phase, we relaxed the bias parameters to obtain less sparse features, which led to better accuracy. The intuition behind this relaxation is that when we learn the features we prefer to enforce a hard threshold to select only good feature representatives and learn from them. But, when the learning is completed, the threshold can be relaxed to also include bad examples of the features. By relaxing the sparsity, the classifier is free to assign its desired weights and thresholds to each of the features.

Gradient descent. During CD learning, we need to update the parameters to minimize the contrastive divergence objective. We performed batch CD gradient descent with an additional momentum from the previous gradient update. We subdivided the training data into batches of roughly 100 examples. Momentum was set to 0.5 for the first few epochs and then changed to 0.9. In learning CRBMs, the value of the learning rate was influential in the features learned. Choosing a large learning rate caused some of the feature maps to a similar pattern, and in fact some of the features were dismissed. Small learning rates also did not perform well since the parameters were not changed that much and some of the learned filters remained noisy. We tested a set of different values for η and selected the one with less noisy and more active features.

Code. We implemented most of our code in the Matlab environment. For the SVM we used an of-the-shelf software called SVM light [18]. For CD training of CRBMs, we implemented our code based on the publicly available code of Hinton & Salakhutdinov for training RBMs [14]. We implemented filtering as a multi-thread C++ code, but if one has a CUDA-enabled graphics card available, in the CUDA framework filtering can be implemented much more efficiently.

5.1 MNIST handwritten digits

MNIST is a popular dataset in the machine learning and vision community. It has $60,000$ training images of 28×28 pixels, each containing an isolated and centered handwritten digit. A test set of $10,000$ digits is also available with similar characteristics. In Fig. 5.1 a selection of the MNIST digits is shown.

We used the full training set of MNIST digits, without considering the digit labels, to train a four layer hierarchical feature detector. First layer filters and the responses of an input zero digit to them are depicted in Fig. 5.2. The third layer larger scale features are visualized in Fig. 5.3. Both of these features were learned automatically using CD learning on two separate CRBMs. After unsupervised learning of the feature extractor, RBF SVMs were trained on top the topmost pooling layer using different amounts of labeled data. We

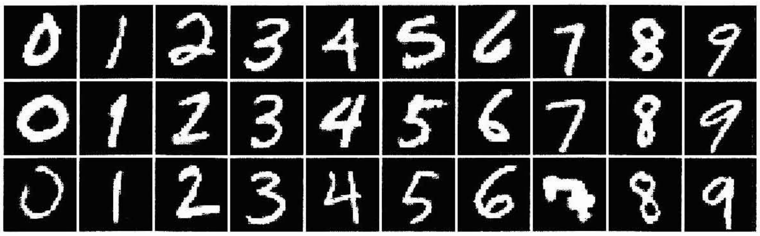


Figure 5.1: A Selection of MNIST digits

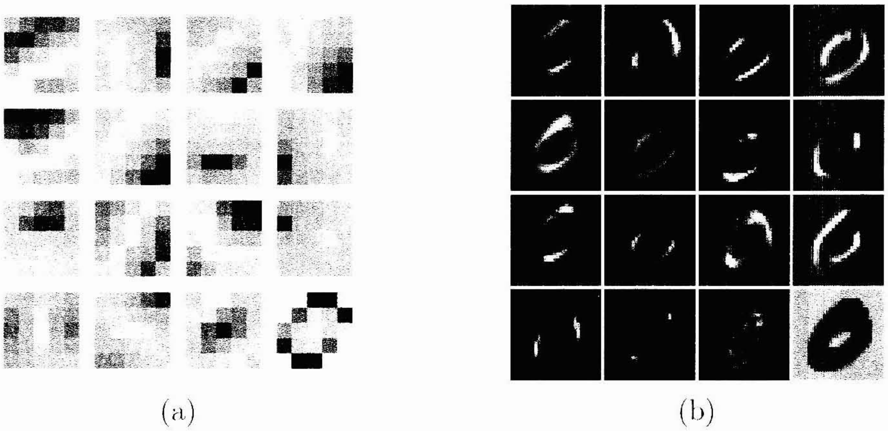


Figure 5.2: First layer 5 × 5 filters obtained by training CRBM on handwritten digit images.

evaluated all these classifiers (trained using different amounts of labeled data) on the original MNIST test set.

Because visualizing 3D filters learned by the second filtering layer is hard, we visualize the patches of digit images that highly responded to the second filtering units. In Fig. 5.3 we visualize such representative patches for a random selection of learned features. Notice how these features really look like the digit parts.

Table 5.1 shows the error rate of our classifier trained on the full set of $60,000$ labeled digits. Our model is significantly superior to RBF SVM on pixels and the fully connected neural network of Hinton et al. [14] that was trained using BP on greedy RBM-style pretraining. This comparison shows that elements such as local feature extraction and pooling are crucial for obtaining good performance on MNIST. Our results are also better than LeNet-5 and the RBF SVM that were trained on LeNet-5's features (see [20]). Lee et al.

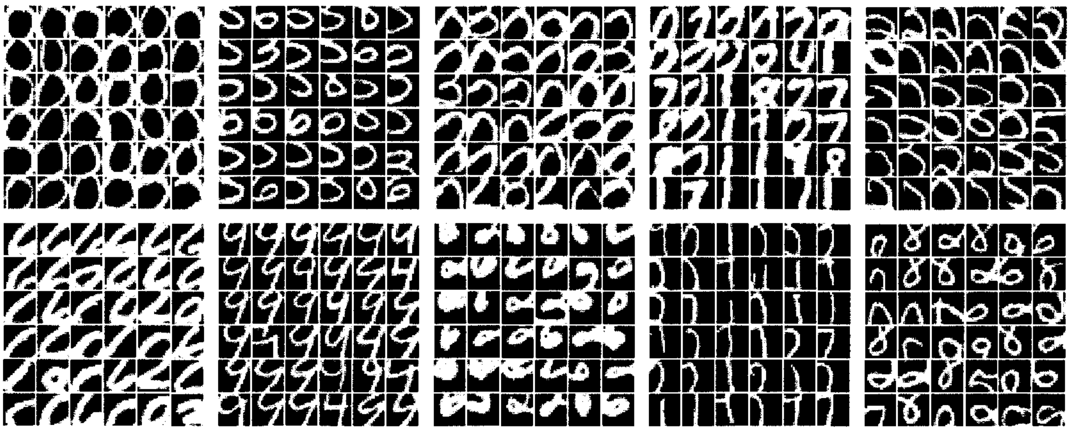


Figure 5.3: Each plate corresponds to a set of 14×14 patches from different digit images that highly activated a third layer hidden unit. The 8 illustrated out of 50 features are selected randomly.

al [23] followed the same approach of stacked CRBMs, but we managed to get better results than this concurrent work. It is hard to directly compare our error rate with those of large CNN models such as [31] and [30], because many details are involved in our and each of those models. The feature size column gives an intuition about the size of the model trained. This value is the size of final feature vector that used for classification. It can be seen that our model is much smaller that the large CNN models. Another difference is that we used RBF kernel SVM for classification but they used multilayer Neural Nets. However the error rates are very close to each other.

Table 5.2 shows the error rate of the final classifier for different amounts of labeled data. We report the result averaged over 10 runs of our classifier over randomly chosen chunks of training data. The model of Ranzato et al. [30] is about 2.5 times larger than ours, and a two layer neural network was used as their final classifier. Although our model delivers slightly higher error rate on the full MNIST task, it is smaller and achieves lower error when fewer labeled training data were accessible. We believe by training larger models we would be able to improve the accuracy for the full MNIST task as well. In table 5.2 notice the difference between our results and pure BP on random initialization. In both our method and [30] the weights were trained in unsupervised fashion and un-labeled data were also incorporated. The BP algorithm cannot benefit from un-labeled data and when labeled

Model	Error	Feature size
RBF SVM on pixels	1.4%	784
Deep Belief Net, RBM pretraining [14]	1.2%	2000
LeNet-5 [21]	0.95%	150
LeNet-5, RBF SVM [20]	0.83%	150
Stacked CRBMs [23]	0.82%	–
Stacked CRBMs, RBF SVM	**0.68%**	1225
Large CNN, unsupervised [30]	0.64%	3200
Large CNN, supervised [31]	0.60%	3200

Table 5.1: MNIST Error rate of different methods when all the labeled training digits were used. The models were not allowed to extend the training set by transforming the images.

Training size	CRBMs	CNN [30]	Pure BP [30]	PoP [2]
20000	**0.90** ±0.05	0.76	0.80	–
10000	**1.11** ±0.07	0.85	0.84	.8
5000	**1.45** ±0.04	1.52	1.98	1.52
2000	**2.07** ±0.10	2.53	3.05	–
1000	**2.63** ±0.13	3.21	4.48	2.14
300	**4.67** ±0.36	7.18	10.63	3

Table 5.2: MNIST error rate as function of training set size. For each row, errors of 10 different SVM runs on different selections of training digits is averaged and reported as our results. The standard deviation of errors is also indicated.

data are small BP performs poorly.

Details The input of our feature extractor is a 32×32 image obtained by evenly zero padding an original 28×28 MNIST digit. As the first layer, we used 15 filters of 5×5 pixels (Fig. 5.2), followed by a sigmoid non-linearity. The first subsampling layer is maximum pooling over 2×2 non-overlapping subwindows. The second filtering layer consists of 3D filters that operate on the feature maps resulted from lower feature detection. Each of the filters has $15 \times 5 \times 5$ parameters that encodes a combination of 5×5 regions of lower level feature maps. We employed 50 of these filters, followed by non-linearity and another 2×2 max pooling. As the final discriminative layer, we combined 10 one-vs-rest binary SVMs, and built a ten-class digit classifier.

To add the desired sparsity to the CRBMs, we set the bias terms of the first and second CRBMs to -6. Without this sparsity the learned filters did not have any visual interpretation. As can be seen in Fig. 5.2 when biases were set to -6, filter responses are almost zero everywhere except at a few locations. In our experiments we observed that having a single non-sparse feature map helps the binary CRBMs and causes the features to become more active and converge better. We added one feature with fixed bias of 0 to the binary CRBMs in both of filtering layers. In Fig. 5.2 the bottom right filter is the non-sparse one, which activates on the background and becomes inactive on the foreground. After training each of the CRBMs we removed the single non-sparse feature because of its different statistics. After learning the filter banks we relaxed the sparsity of features by reducing their fixed bias term. We did cross validation on small subsets of training set and found -1.2 to be the best bias for discrimination. This value was chosen from 5 candidates evenly spread between 0 and -6.

Classifier. As mentioned above, we followed the approach of one-vs-rest classifiers for multi-class classification. Thus, 10 RBF kernel SVMs were trained to separate each of the digit classes from the others. The bandwidth parameter of RBF kernel, denoted by g^*, was set according to the following rule of thumb:

$$g^* = 1/mean\left(||x_i - x_j||_2^2\right) \tag{5.1}$$

where we take the mean over all the data pairs. The regularizer parameters C of the SVMs were tuned also heuristically based on a similar rule of thumb:

$$C^* = 1/mean\left(\exp\{-g^*||x_i - x_j||_2^2\}\right). \tag{5.2}$$

Doing cross validation is time consuming but improves some of the results a bit. We did five fold cross validation for small training sets and its final accuracy was not very different.

5.2 INRIA pedestrian detection benchmark

The INRIA person dataset is a challenging benchmark for pedestrian detection. This dataset includes 2416 positive training bounding boxes of size 128×64, and 1218 negative training images of different sizes. It includes 1132 positive test bounding boxes of size 128×64, and 453 negative test images. This dataset involves extreme illumination conditions, occlusion, background clutter, and a variety of human poses. Results on INRIA often are reported as miss rate at different false positive per detection window (FPPW) rates.

In contrary to the digit recognition task where the background was a simple black layer, background clutter is an element of the INRIA images. Therefore, in addition to our part-like features, we need to have a template for the whole human figure. This template helps the model to rule out images containing spurious parts, and achieve more accurate results. We combine our features with HOG descriptors, which are about three times smaller than our high level feature. Adding the HOG descriptors helps the SVM classifier to create a more detailed human figure template. Further, it enhances the robustness of the model against illumination change because the HOG descriptors are locally contrast normalized.

Our feature learning was only trained on the INRIA positive set (human images), to make the model able to extract human part features. We learned 15 filters of 7×7 pixels, depicted in Fig. 5.4, as the first layer. A continuous CRBM is trained on the contrast normalized 32×32 grayscale image patches to obtain these filters. After a max pooling layer with 4×4 subsampling window, we trained a binary CRBM on top of the lower level feature maps extracted from full images. We learned 30 3D filters of size $15 \times 5 \times 5$ for the third layer, six of them were illustrated in Fig. 1.2. The last max pooling window size was set to 2×2. We set the bias parameter of the first CRBM (continuous one) to -4 and did not include any non-sparse feature map. However, for the second CRBM (binary one) we included a non-sparse feature map with bias of zero and set the other biase terms to -6 just like the digit experiments.

The final outputs of stacked CRBMs (high level features) were concatenated with HOG descriptors into a feature vector. To combine the features we had to make the range of both feature types roughly the same to equalize the effect of SVM regularizer on both of them.

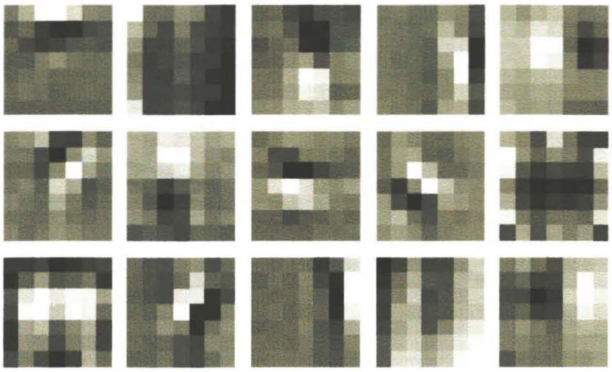


Figure 5.4: First layer 7×7 filters learned from INRIA positive training set.

Since all features are bounded from below by zero, we only scaled the features to make their variance identical over the whole training data, and learned a linear SVM on top of their combination.

Fig. 5.5 plots the miss rate (number of true pedestrians that we missed) as a function of FPPW (number of non-human windows that we incorrectly labeled as human). Clearly, we want to achieve lower miss rate while the FPPW is kept low. We can see a significant improvement over HOG results when our features were concatenated with them. Also we compare our results with another state-of-the-art approach by Tuzel et al. [37]. Our result achives similar accuracy.

We end this chapter by visualizing the hardest INRIA test examples according to our classifier. In Fig. 5.6 both human and non-human images that highly confused our model are shown. Some of the human images are really hard e.g., those that are in unusual poses or in extreme illumination conditions. Some of the negative examples are highly cluttered and have many directed edges. Hard negative examples reveal that our classifier puts high weights on the parallel vertical lines in two sides of the image windows.

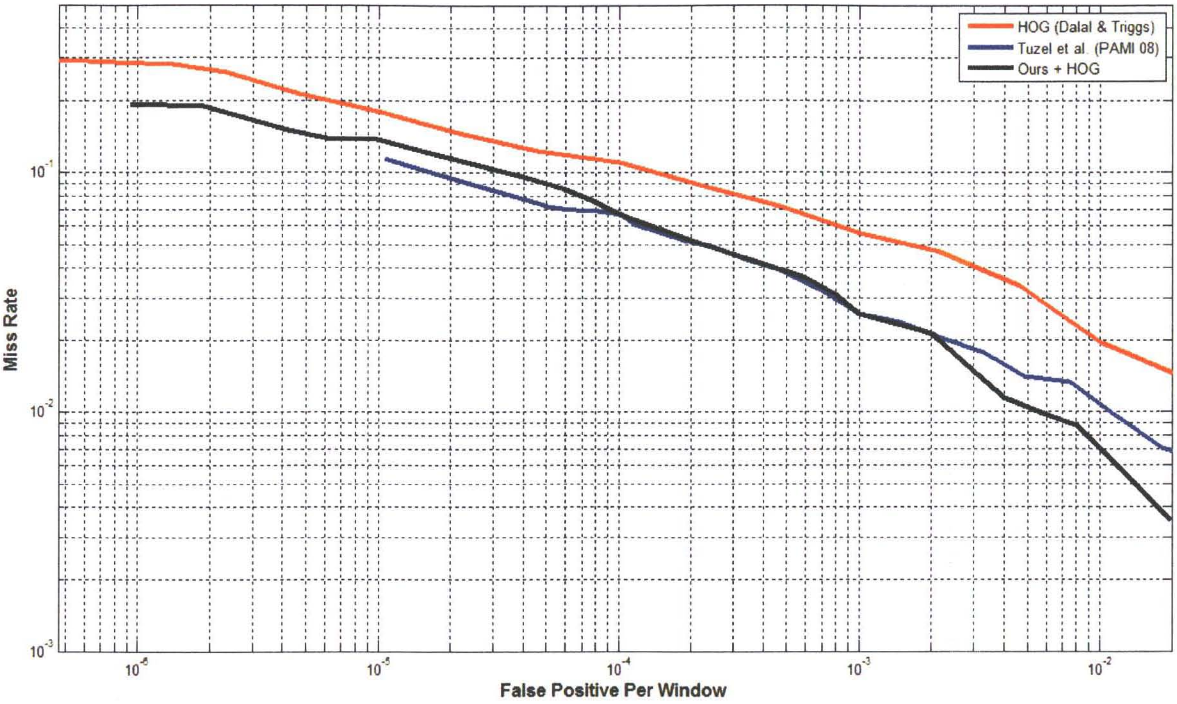


Figure 5.5: INRIA DET curves for HOG features, the combination of HOG and the CRBM features, and Tuzel et al. [37] method (Best viewed in color).

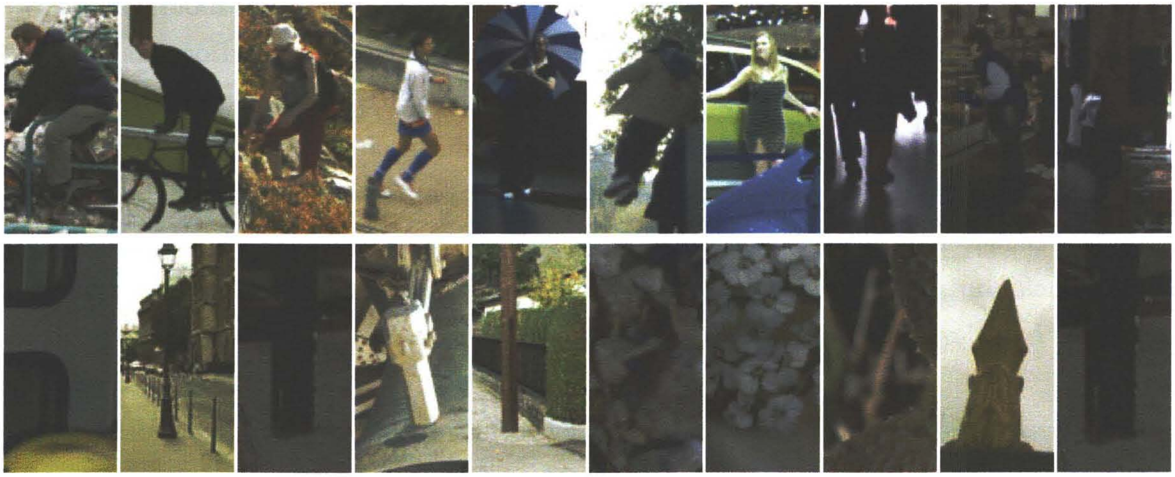


Figure 5.6: INRIA test images that were hardest for our model to classify. Top row shows the human examples with lowest scores, and bottom row illustrate the negative examples with highest scores.

Chapter 6

Conclusion

In this thesis, we have described a layerwise algorithm for learning hierarchical structures of local features. This algorithm learns generic features at the bottom levels and features specific to an object class in the top layers. The algorithm extends the Restricted Boltzmann Machine model by introducing weight sharing to define features that are replicated over spatial neighborhoods. By using this Convolutional Restricted Boltzmann Machine (CRBM) to model the distribution of a set of images, we learn a set of features which are tuned to represent a particular object class. These features are tested on the MNIST handwritten digits and INRIA pedestrian detection benchmark and obtain results comparable to state-of-the-art methods on both tasks.

The main motivation behind our layerwise learning algorithm was to address the limitations of backpropagation (BP) algorithm. BP requires careful initialization of weights and it might get trapped in local minimas with the result that the performance is good on the training but poor on the test set. The layerwise approach that we presented does not have these problems and leads to good performance on realistic vision problems such as pedestrian detection. We have adopted the layerwise learning for local feature hierarchies that might have applications in other fields such as sound and speech analysis and natural language processing.

In addition, our proposed algorithm benefits from un-labeled data. This is a very important characteristic because in many problems labeled data are scarce. Our framework provides a mechanism for using un-labeled data in learning features and later injecting the labels for training a classifier. The capability of stacked CRBMs in making use of un-labeled should be further explored for tasks that lacks large labeled datasets.

Appendix A

Basic Operations

A.1 Filtering

Here we describe the filtering operation in more detail. Consider a 2D matrix of numbers representing a grayscale image. For extracting local features from different locations of the image, a standard procedure is to perform *filtering* using a *filter kernel.* A filter kernel (filter for short) is another 2D matrix of numbers but much smaller than the original image e.g., 3×3 such a those visualized in Fig. 1.1. One can think of a filter kernels as an interesting spatially local visual feature that we want to extract from the images. For extracting a particular feature we take the corresponding filter kernel and correlate it with different neighborhoods of the image and store the results in a filter response matrix–this is called Filtering.

If the image is 3×3 and the filter is also 3×3, then the filtering response will be only one number: the sum of element-wise product of the 3×3 image and the 3×3 filter (same as the dot product of them). If the image is larger than 3×3, then we place the filter over all the 3×3 neighborhoods of the image and take their dot product. All the dot product responses extracted from different 3×3 neighborhoods would be stored regarding their location in a filtering response matrix.

For example if the image is 4×4, it has $4(=2\times2)$ overlapping 3×3 neighborhoods. Therefore the filter response will be a 2×2 matrix of the dot product responses. The filter response matrix represents the spatially local feature responses extracted from different locations of the input image.

To be more quantitative, consider an image:

$$\begin{bmatrix} 5 & 5 & 5 & 5 \\ 1 & 3 & 2 & 5 \\ 1 & 2 & 4 & 6 \\ 5 & 5 & 5 & 5 \end{bmatrix}$$

and the filter kernel:

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Filtering the above image with the above filter induces the following filter response:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

A.2 Pooling

Here we give a quantitative example of the pooling operation. Assume we want to perform 2×2 max pooling over non-overlapping regions and we are given the following matrix as the input:

$$\begin{bmatrix} 4 & 6 & 2 & 8 \\ 2 & 7 & 1 & 7 \\ 3 & 1 & 8 & 9 \\ 1 & 2 & 7 & 10 \end{bmatrix}$$

We divide this matrix into 4 sub-matrices each having 2×2 elements as illustrated below. Next, from each sub-matrix we select the maximum value and place these maximum terms in the output matrix:

$$\left[\begin{array}{cc|cc} 4 & 6 & 2 & 8 \\ 2 & 7 & 1 & 7 \\ - & - & - & - \\ 3 & 1 & 8 & 9 \\ 1 & 2 & 7 & 10 \end{array}\right] \longrightarrow \begin{bmatrix} 7 & 8 \\ 3 & 10 \end{bmatrix}$$

Bibliography

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

[2] Y. Amit and A. Trouvé. Pop: Patchwork of parts models for object recognition. Technical Report 2, 2007.

[3] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, and U. Montreal. Greedy layer-wise training of deep networks. In *Advances in Neural Info. Processing Systems*, page 153. The MIT Press, 2007.

[4] M. Carandini, J.B. Demb, V. Mante, D.J. Tolhurst, Y. Dan, B.A. Olshausen, J.L. Gallant, and N.C. Rust. Do we know what the early visual system does? *Journal of Neuroscience*, 25(46):10577–10597, 2005.

[5] C. C. Chang and C. J. Lin. *LIBSVM: a library for support vector machines*, 2001.

[6] H. Chen and A. F. Murray. Continuous restricted boltzmann machine with an implementable training algorithm. *IEEE Proc. Vision, Image and Signal Processing*, 150(3):153–159, 2003.

[7] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. IEEE Conference Computer Vision and Pattern Recognition*, 2005.

[8] G. Desjardins and Y. Bengio. Empirical evaluation of convolutional RBMs for vision. Technical report, Tech Report, 2008.

[9] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[10] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800, 2002.

[11] G. E. Hinton. "deep belief networks". *Scholarpedia*, 4(5):5947, 2009.

[12] G. E. Hinton, P. Dayan, BJ Frey, and RM Neal. The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

[13] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, 2006.

[14] G. E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

[15] G.E. Hinton. To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535, 2007.

[16] DH Hubel and TN Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106, 1962.

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

[18] T. Joachims. Making large scale SVM learning practical. *Advances in Kernel Methods - Support Vector Learning*, 1999.

[19] H. Larochelle and Y. Bengio. Classification using discriminative restricted boltzmann machines. In *Proc. of the International Conference on Machine Learning*, 2008.

[20] F. Lauer, C. Y. Suen, and G. Bloch. A trainable feature extractor for handwritten digit recognition. *Pattern Recognition*, 40(6):1816–1824, 2007.

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[22] H. Lee, C. Ekanadham, and A. Ng. Sparse deep belief net model for visual area v2. In *Advances in Neural Info. Processing Systems*, pages 873–880, 2007.

[23] H. Lee, R. Grosse, R. Ranganath, and A.Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proc. International Conference on Machine Learning*. ACM New York, NY, USA, 2009.

[24] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision*, 60(2):91–110, 2004.

[25] J. Mutch and D. G. Lowe. Multiclass object recognition with sparse, localized features. In *Proc. IEEE Conference Computer Vision and Pattern Recognition*, 2006.

[26] Mohammad Norouzi, Mani Ranjbar, and Greg Mori. Stacks of convolutional restricted boltzmann machines for shift-invariant feature learning. In *Proc. IEEE Conference Computer Vision and Pattern Recognition*, 2009.

[27] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[28] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision research*, 37(23):3311–3325, 1997.

[29] N. Pinto, D.D. Cox, and J.J. DiCarlo. Why is real-world visual object recognition hard. *PLoS Computational Biology*, 4(1):e27, 2008.

[30] M. Ranzato, F. J. Huang, Y. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proc. IEEE Conference Computer Vision and Pattern Recognition*, 2007.

[31] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In J. Platt et al., editor, *Advances in Neural Info. Processing Systems.* MIT Press, 2006.

[32] S. Roth and M. J. Black. Fields of experts: A framework for learning image priors. In *Proc. IEEE Conference Computer Vision and Pattern Recognition*, 2005.

[33] D.E. Rumelhart, G. E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[34] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio. Robust object recognition with cortex-like mechanisms. *IEEE Trans. PAMI*, 29(3):411–426, 2007.

[35] P. Smolensky. *Parallel distributed processing: explorations in the microstructure of cognition*, volume 1: foundations, chapter Information processing in dynamical systems: foundations of harmony theory, pages 194–281. MIT Press, Cambridge, MA, USA, 1986.

[36] Y. W. Teh, M. Welling, S. Osindero, and G. E. Hinton. Energy-based models for sparse overcomplete representations. *The Journal of Machine Learning Research*, 4:1235–1260, 2003.

[37] O. Tuzel, F. Porikli, and P. Meer. Pedestrian detection via classification on riemannian manifolds. *IEEE Trans. PAMI*, 30:1713–1727, 2008.

[38] P. Viola and M. J. Jones. Robust real-time face detection. *Int. Journal of Computer Vision*, 57(2):137–154, 2004.

[39] Y. Weiss and W. T. Freeman. What makes a good model of natural images? In *Proc. IEEE Conference Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[40] M. Welling, G. E. Hinton, and S. Osindero. Learning sparse topographic representations with products of student-t distributions. In *Advances in Neural Info. Processing Systems*, pages 1383–1390, 2003.