# Learning Feature Hierarchies for Object Recognition

by

Koray Kavukcuoglu

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

January 2011

Professor Yann LeCun

© Koray Kavukcuoglu

All rights reserved 2011.

To my family.

# Acknowledgements

I would like to thank my advisor Prof. Yann LeCun for his continuous guidance during my thesis work. It has been a great privilege to work with and learn from him. I would also like to thank Dr. Ronan Collobert and Dr. Jason Weston for providing me the opportunity to work with them. I am grateful to the members of my thesis committee for their time and comments to improve this thesis. Finally I would like to mention that the discussions and collaborations at Prof. Yann LeCun's lab have contributed a lot to this PhD thesis.

Most importantly, I am indebted to my wife, daughters and all my family, without their endless support I would not be able to complete this PhD work.

# Abstract

In this thesis we study unsupervised learning algorithms for training feature extractors and building deep learning models. We propose sparse-modeling algorithms as the foundation for unsupervised feature extraction systems. To reduce the cost of the inference process required to obtain the optimal sparse code, we model a feed-forward function that is trained to predict this optimal sparse code. Using an efficient predictor function enables the use of sparse coding in hierarchical models for object recognition. We demonstrate the performance of the developed system on several recognition tasks, including object recognition, handwritten digit classification and pedestrian detection. Robustness to noise or small variations in the input is a very desirable property for a feature extraction algorithm. In order to train locally-invariant feature extractors in an unsupervised manner, we use group sparsity criteria that promote similarity between the dictionary elements within a group. This model produces locally-invariant representations under small perturbations of the input, thus improving the robustness of the features. Many sparse modeling algorithms are trained on small image patches that are the same size as the dictionary elements. This forces the system to learn multiple shifted versions of each dictionary element. However, when used convolutionally over large images to extract features, these models produce very redundant representations. To avoid this problem, we propose convolutional sparse coding algorithms that yield a richer set of dictionary elements, reduce the redundancy of the representation and improve recognition performance.

# Contents

# List of Figures

# List of Tables

# Nomenclature

| | |
|---|---|
| $\mathcal{Z}$ | Set of codes for each input sample |
| $z$ | Representation (code) |
| $\mathcal{D}$ | Dictionary matrix |
| $\mathcal{F}_e$ | Predictor function |
| $\mathcal{X}$ | Input dataset |
| $x$ | Input sample |
| $\lambda$ | Sparsity penalty coefficient |
| $F_{CSG}$ | Filterbank layer composed of a set of convolutional kernels, followed by a tanh non-linearity and gain coefficients |
| $g$ | Predictor gain parameters, one component per each feature map |
| $K$ | Predictor Parameters |
| $k$ | Predictor filtering parameters |

| | |
|---|---|
| $N$ | Local contrast normalization layer |
| $P$ | Set of neighborhood (pool) definitions |
| $P_A$ | Average pooling and sub-sampling layer |
| $P_i$ | $i^th$ neighborhood (pool) definition |
| $P_M$ | Max pooling and sub-sampling layer |
| $R$ | A single stage feature extractor with weights initialized randomly |
| $R^+$ | A single stage feature extractor with randomly initialized weights, followed by supervised fine-tuning |
| $R^+R^+$ | A two stage feature extractor with randomly initialized weights, followed by supervised fine-tuning |
| $R_{abs}$ | Absolute value rectification layer |
| $RR$ | A two stage feature extractor with weights initialized randomly |
| $U$ | A single stage feature extractor with weights initialized using PSD |
| $U^+$ | A single stage feature extractor with weights initialized using PSD, followed by supervised fine-tuning |
| $U^+U^+$ | A two stage feature extractor with weights initialized using PSD, followed by supervised fine-tuning |
| $UU$ | A two stage feature extractor with weights initialized using PSD |

| | |
|---|---|
| $w$ | Gaussian weighting function |
| PMK-SVM | Pyramid Match Kernel SVM Classifier (Lazebnik et al., 2006) |

# Chapter 1

# Introduction

In this thesis we propose using hierarchical models with multiple stages of trainable feature extractors (LeCun et al., 1998a, 2004; Serre et al., 2005; Ranzato et al., 2007b; Larochelle et al., 2009) for object recognition. By adapting feature extractors to the data at hand, we aim to develop a common architecture that can be used on different domains. We also propose unsupervised learning methods for training the stages of the hierarchical model in order to benefit from abundant unlabeled data available in many domains. Each stage of feature extractor is composed of a filter-bank layer, a non-linear transformation layer and a pooling layer. A multi-stage hierarchical model stacks several of such stages where the output of a stage is directly provided as input to the successive one. Figure 1.1 shows a single stage feature extractor. Convolutional neural networks (ConvNets) (LeCun et al., 1998a), HMAX model (Serre et al., 2005) and commonly used feature extractors like SIFT (Lowe, 2004) and Histogram of Gradients (HoG) (Dalal and Triggs, 2005)

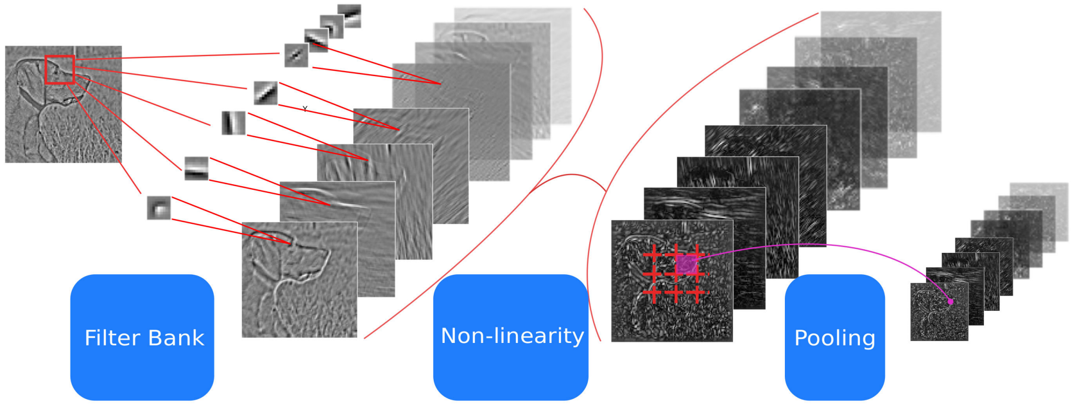

Figure 1.1: A single stage of generic feature extraction model that first applies a filter-bank to the input, followed by one or more non-linear functions and finally a pooling operation that provides invariance to local transformations.

also share this same framework. ConvNets train the filter-bank layers, followed by a sigmoidal non-linear function and trained weighted average pooling function, HMAX on the other hand, uses Gabor functions in filter-bank layer, followed by max pooling operation.

Deep architectures can extract gradually more abstract and invariant representations through multiple levels (Ranzato et al., 2007a; Goodfellow et al., 2009). Training a deep multi-stage neural network is difficult in general because the problem is very non-linear and non-convex. There have been successes though, for example convolutional neural networks (LeCun et al., 1998a) which use weight sharing to exploit translational invariance in images. Another successful approach described in (Hinton and Salakhutdinov, 2006) shows that layer wise unsupervised training can be used to initialize the parameters of a deep network which improves the performance significantly. Several authors have also demonstrated that layer wise unsupervised or semi-supervised learning can be used to train deep architectures (Hinton and Salakhutdinov, 2006; Bengio et al., 2007; Larochelle et al., 2009; Ranzato et al., 2006, 2007a; Weston et al., 2008; Collobert and Weston, 2008; Kavukcuoglu et al., 2010; Qi et al., 2009; Ahmed et al., 2008). Ideally, such an algorithm should be able to explain its data well with a hidden representation that is more abstract and less correlated than the input. This hidden representation can be used as input to the subsequent layers of the architecture.

On the other hand, several successful human-engineered feature extractors for object recognition have been introduced (Lowe, 2004; Dalal and Triggs, 2005). These systems share a common architecture composed of a filter bank (generally using oriented edge detectors), an often sparsifying non-linear operation such as quantization, winner-take-all, normalization or saturation and a pooling operation that combines nearby values in either spatial or feature space. A standard architecture for an object classification system is obtained by extracting features on a dense, regularly spaced grid over an image followed by a supervised classifier such as an SVM (Lazebnik et al., 2006; Grauman and Darrell, 2005; Mutch and Lowe, 2006). Despite being successful on tasks that contain natural images, these methods perform rather poorly on different types of inputs due to their domain specific feature extraction stage. Moreover, (Boureau et al., 2010) has shown that a careful selection of the number of orientations, pooling regions, and the granularity of the dense feature extraction can have significant effect on the recognition performance, however, these considerations are often overlooked.

The models we propose also share the common components explained above, however, they are adapted to the input data and can be used in a hierarchy. In order for these methods to be applicable on large datasets, it is important to have an efficient algorithm for computing the hidden representation. For that purpose, we have developed an unsupervised learning algorithm with efficient inference based on well known sparse coding methods, named *Predictive Sparse Decomposition* (PSD) (Kavukcuoglu et al.). Sparse coding methods deal with finding the sparsest representation of a signal while achieving best possible reconstruction using a given overcomplete dictionary.

It is well established that sparse modeling algorithms can learn dictionaries that represent natural signals (image, audio, ...) quite successfully (Olshausen and Field, 1997, 1996; Olshausen, 2000). However, for any new input signal, inference in sparse coding algorithms requires a possibly expensive optimization process and although many efficient algorithms have been developed (Efron et al., 2004; Mallat and Zhang, 1993; Chen et al., 1998; Lee et al., 2007; Aharon et al., 2006; Mairal et al., 2008; Murray and Kreutz-Delgado, 2006; Rozell et al., 2008; Beck and Teboulle, 2009; Li and Osher), the solution procedures are still too slow for real-time applications and large-scale object recognition tasks. In addition, sparse representations can change drastically under small perturbations of the input data and may not be suitable for object recognition (Kavukcuoglu et al., 2009; Rozell et al., 2008). Another important problem with sparse coding methods is that they are trained using local patches or parts and when applied on large inputs convolutionally for feature extraction, they produce highly redundant representations.

In this thesis we describe four variants of predictive sparse coding that address these issues. In the simplest, originally introduced in PSD, we constructed a smooth regressor function that can approximately predict optimal feature representations in a feed-forward manner. We also developed a more principled approach for achieving stable sparse representations via a variational marginalization approach that can produce more stable representations than sparse coding solution. We also describe an extension to PSD that produces locally invariant representations (Kavukcuoglu et al., 2009), by building upon the idea of complex cells which was proposed in (Hyvarinen and Koster, 2007), (Osindero et al., 2006; Welling et al., 2003). In this case, the sparsity penalty is applied on groups of units rather than each single unit (Kavukcuoglu et al.). Using this modified formulation and again training a predictor function, we have obtained a feed-forward, locally invariant feature extractor system named *Invariant Predictive Sparse Decomposition* (IPSD). Since IPSD is adapted to data it performs similarly to well known SIFT (Lowe, 2004) features on tasks that use natural images and better than SIFT on tasks that SIFT is not well suited (like MNIST digit recognition). This work has shown that it is possible to learn pooling functions in an unsupervised manner. Finally, we introduce a version of PSD that can be trained on large images convolutionally, producing less redundant dictionaries and improving recognition performance.

Following the unsupervised layer-wise initialization procedure proposed in (Hinton and Salakhutdinov, 2006), we have built a multi-layer object recognition system (Jarrett et al., 2009). We have shown that using appropriate non-linearities can have very significant affect on the performance of multi-stage architectures. In a multi-layer architecture, the most crucial components were found to be absolute value rectification and *Local Contrast Normalization* (LCN) (Schwartz and Simoncelli, 2001; Wainwright et al., 2002; Pinto et al., 2008).

In chapter 2 we explain the unsupervised learning model based on sparse coding with efficient inference using a predictor function. Chapter 3 introduces the invariant formulation of the PSD method and chapter 4 explains the importance of the non-linear functions in a multi-stage system. In chapter 5, we explain the convolutional sparse coding model and its advantages as a feature extraction system for recognition. Finally, in chapter 6 we explain the variational marginalization method for sparse coding, followed by conclusion and future work.

## 1.1 Engineered Feature Extraction Systems

The widely used SIFT descriptor of (Lowe, 2004) uses the same basic design explained above. First oriented edge detectors at several (usually 8) orientations are convolved with an image, followed by a contrast enhancement operation that takes each response value to the $p^{th}$ power ( (Lazebnik et al., 2006) uses $p = 9$) and then a weighted summation over a local region ($4 \times 4$ pixels). The resulting 8 dimensional histograms are concatenated over 16 pooling regions on a $4 \times 4$ grid to produce 128 dimensional SIFT features which are finally normalized to unit L2 norm.

Another popular feature extraction method is the Histogram of Gradients (HoG) method (Dalal and Triggs, 2005). In this method, the gradient computation is performed using two centered edge filters, $[-1, 0, 1]$ and its transpose, followed by the soft binning operation performed at each pixel which accumulates the magnitude of the gradient that falls into one of 9 orientation bins discarding the sign information (effectively 18 bins). Finally each local region (or a block), (generally between $6 \times 6$ to $8 \times 8$ pixels) is normalized by $\ell^2$ norm of the descriptor. The blocks are normalized independently but overlap by half of the size of the block and the final descriptor is obtained by concatenating the individual descriptors on a $3 \times 3$ grid.

## 1.2 Unsupervised Training of Sparse Models

In linear generative models for images, each image ($x$) is represented as linear combination of several basis functions (or columns of a dictionary matrix) ($\mathcal{D}_i$) using mixing coefficients ($z_i$) and the aim is to infer the representation $z$ given an input $x$ and dictionary $\mathcal{D}$.

$$x = \sum_i \mathcal{D}_i z_i \tag{1.1}$$

If the number of dictionary elements is the same as the size of the input, then one can obtain the representation ($z$) by applying the inverse of the dictionary matrix to the input.

$$z = \mathcal{D}^{-1}x \tag{1.2}$$

In sparse coding models, the dictionary $\mathcal{D}$ is overcomplete (the number of dictionary elements is larger than the dimensionality of the input), thus there are infinitely many solutions for $z$ and a sparsity regularization term on the representation $z$ is used to obtain a unique solution. Different such models have been proposed in the literature, and one can generally represent the problem as a compound energy function:

$$E_{unsup} = \mathcal{M}(x, \mathcal{D}z) + s(z) \tag{1.3}$$

where $\mathcal{M}$ measures the reconstruction accuracy of the model and $s$ measures the sparsity of the representation $z$. The almost unanimous choice for the reconstruction measure is the squared $\ell^2$ norm of the difference between input and the model reconstruction $\|x - \mathcal{D}z\|_2^2$. The sparsity cost $s$ on the other hand can take many different forms including $\ell^0$ norm (Mallat and Zhang, 1993) ($s(z) = \lambda|z|_0$) which is the number of non-zero elements in $z$, $\ell^1$ norm (Daubechies et al., 2004; Beck and Teboulle, 2009) ($s(z) = \lambda|z|_1$), which is the sum of the absolute values of all elements, or some other form that encourage sparse representations like $s(z) = \log(1 + x^2)$.

More important are the models that adopt the dictionary matrix to the data.

Independent Component Analysis (ICA) (Comon, 1994) model uses an orthonormal dictionary matrix, therefore once the model is trained, inferring the representation $z$ requires only a matrix-vector multiplication of the input $(x)$ with the inverse of the dictionary matrix $(\mathcal{D}^{-1})$. (Olshausen and Field, 1997) is one of the first models that used sparse coding to adopt an overcomplete dictionary to the data using $\ell^1$ norm regularization on $z$. Learning the parameters of the model, that is the dictionary matrix $\mathcal{D}$ can be done using either online or batch methods. A common framework for online learning which is used in (Olshausen and Field, 1997; Ranzato et al., 2006, 2007a; Mairal et al., 2009) consists of two main operations: **1.** Minimize $E_{unsup}$ to obtain the optimal representation $z^*$. **2.** Keeping $z$ fixed, update the dictionary matrix $\mathcal{D}$.

However inference in sparse coding is considerably costly, since it requires an iterative optimization process. To overcome this problem, the idea of using a feed-forward regressor function for predicting sparse representations was first introduced in (Ranzato et al., 2006). In that work a temporal soft-max function was used to enforce sparsity on activation of each component of the representation over time (or equivalently over a series of input samples) as opposed to the $\ell^1$ penalty used in sparse modeling that penalizes activations of all code units for a given sample. In this model, sparsity of the representation was enforced through the following per component function:

$$\bar{z}_i(k) = \frac{\eta e^{\beta z_i(k)}}{\zeta_i(k)} \tag{1.4}$$

$$\zeta_i(k) = \eta e^{\beta z_i(k)} + (1-\eta)\zeta_i(k-1) \tag{1.5}$$

The sparsity cost was not explicitly included in the energy function, however the sparsifying logistic was used as part of the reconstruction process. The predictor function was a linear projection of the input with a matrix $W_C$.

$$E = \frac{1}{2}\|x - \mathcal{D}\bar{z}\|_2^2 + \frac{1}{2}\|z - W_C x\|_2^2 \tag{1.6}$$

The learning was performed using the two step process explained above, however not only the dictionary matrix $\mathcal{D}$, but also the predictor matrix $W_C$ was updated.

Following this work, (Ranzato et al., 2007a) introduced the Sparse Encoding Symmetric Machine (SESM), which used a per sample penalty function, and constrained the encoder and decoder to be the transpose of each other. In this work, a fixed logistic function is used to obtain non-negative representation $z$, and the sparsity regularization was enforced through

$$s(z) = \sum_i \log\left(1 + \left[\frac{1}{1 + e^{-gz_i}}\right]^2\right) \tag{1.7}$$

The total energy minimized in this model contains three terms, the reconstruction error, prediction error and the sparsity cost

$$E = \frac{1}{2}\|x - \mathcal{D}l(z)\|_2^2 + \alpha_e\|z - W_C x\|_2^2 + \alpha_s s(z) \tag{1.8}$$

$$l(z) = \frac{1}{1 + \exp(-gz)} \tag{1.9}$$

The learning process for this model is the same as the learning process in (Ranzato et al., 2006) using stochastic gradient descent.

# Chapter 2

# Sparse Modeling for Unsupervised Training

Sparse coding algorithms represent an input signal $x \in \mathcal{R}^m$ using a linear combination of the columns of an overcomplete dictionary matrix $\mathcal{D} \in \mathcal{R}^{m \times n}$, with coefficients $z \in \mathcal{R}^n$ $(n > m)$. In the original sparse coding formulation, the problem is stated as:

$$\min ||z||_0 \ \text{ s.t. } \ x = \mathcal{D}z \tag{2.1}$$

where the $\ell^0$ norm is defined as the number of non-zero elements in a given vector. Unfortunately, the solution to this problem requires a combinatorial search, intractable in high-dimensional spaces. Matching Pursuit methods (Mallat and Zhang, 1993) offer a greedy approximation to this problem. Another way to approximate this problem is to make a convex relaxation by turning the $\ell^0$ norm into an $\ell^1$ norm (Chen et al., 1998). This problem, dubbed *"Basis Pursuit"* in the signal processing community, has been shown to give the same solution to eq. (2.1), provided that the solution is sparse enough (Donoho and Elad, 2003). Since this linear system is under-determined, a sparsity constraint is added that prefers most of the coefficients to be zero for any given input. Furthermore, the problem can be written as an unconstrained optimization problem, named *"Basis Pursuit Denoising"* in (Chen et al., 1998):

$$E_{BPDN} = \min \frac{1}{2}||x - \mathcal{D}z||_2^2 + \lambda \sum_i |z_i| \tag{2.2}$$

This particular formulation, can be seen as minimizing an objective that penalizes the reconstruction error using a linear basis set and the sparsity of the corresponding representation. Many recent works have studied this problem given in eq. (2.2) (Efron et al., 2004; Murray and Kreutz-Delgado, 2006; Beck and Teboulle, 2009; Li and Osher), also extending to the case when the dictionary $\mathcal{D}$ is learned, thus adapting to the statistics of the input. Yet, inference requires running some sort of iterative minimization algorithm that is always computationally expensive.

Matching pursuit (Mallat and Zhang, 1993) provide a solution for the $\ell^0$ problem given in eq. (2.1). It is a greedy method thus it does not find the global minimum of the objective. On the other hand, matching pursuit is a very intuitive method, which gradually minimizes the residual reconstruction error by greedly selecting the best matching dictionary element as given in algorithm 1.

One of the most popular approaches for solving the $\ell^1$ problem given in eq. (2.2) is *Iterative Shrinkage and Thresholding ALgorithm* (ISTA) (Daubechies et al.,

**Algorithm 1** Matching Pursuit (Mallat and Zhang, 1993)

```
function MP($x, \mathcal{D}, nmax$)
    Input: $x$ : input, $\mathcal{D}$ : dictionary, $nmax$ : number of active elements
    Output: $z$
    Initialize: $R = x,\ z = 0,\ n = 0$
    repeat
        $i = \max_j \mathcal{D}_j^T R$
        $z_i \leftarrow \mathcal{D}_i^T R$
        $R \leftarrow R - code_i \mathcal{D}_i$
        $n \leftarrow n + 1$
    until $n = nmax$
end function
```

2004) and particularly the faster version *Fast ISTA* (FISTA) (Beck and Teboulle, 2009). The main operation in these algorithms is successive application of the soft-thresholding operation $h_\lambda(x) = sgn(x)(|x| - \lambda)$. (Beck and Teboulle, 2009) provides a derivation for ISTA methods in terms of gradient minimization of a compound loss function that consists of a smooth convex part and non-smooth convex part. Additonally it provides an improved step size calculation method (Nesterov, 2007). The FISTA method is shown in algorithm 2.

**Algorithm 2** Fast Iterative Shrinkage Thresholding Algorithm (Beck and Teboulle, 2009).

```
function FISTA($x, \mathcal{D}, \lambda, \eta$)
    Input: $x$ : input, $\mathcal{D}$ : dictionary, $\lambda$ : sparsity penalty coefficient, $\eta$ : step size
    Output: $z$
    Initialize: $k = 1,\ z_0 \in \mathbb{R}^n,\ t_1 = 1$
    repeat
        $z_k \leftarrow h_{\lambda * \eta}(z_{k-1} - \eta(\mathcal{D}^T(\mathcal{D} z_{k-1} - x)))$
        if Fast ISTA then
            $t_{k+1} \leftarrow \frac{1+\sqrt{1+4t_k^2}}{2}$
            $z_k \leftarrow z_k + \left(\frac{t_k - 1}{t_{k+1}}\right)(z_k - z_{k-1})$
        end if
        $k \leftarrow k + 1$
    until Change in energy is less than a threshold
end function
```

(Olshausen and Field, 1997) is most probably the first work which extends eq. (2.2) to adopt the dictionary elements (sparse modeling). The basic idea is to minimize the same objective in eq. (2.2) alternatively over coefficients $z$ for a given dictionary $\mathcal{D}$, and then over $\mathcal{D}$ for a given set of $z$ (Aharon et al., 2005; Lee et al., 2007; Ranzato et al., 2006, 2007a; Mairal et al., 2008; Rozell et al., 2008). Note that each column of $\mathcal{D}$ is required to be unit $\ell^2$ norm (or bounded norm) in order to avoid trivial solutions that are due to the ambiguity of the linear reconstruction (for instance, the objective can be decreased by respectively dividing and multiplying $z$ and $\mathcal{D}$ by a constant factor greater than one).

It is well established that sparse modeling algorithms applied to natural images learn basis functions that are localized oriented edges and resemble the receptive fields of simple cells in area V1 of the mammalian visual cortex (Olshausen and Field, 1997). These methods produce feature representations that are sparse, but not invariant. If the input pattern is slightly distorted, the representation may change drastically. Moreover, these features represent information about local texture, and hence, are rather inefficient when used to pre-process whole images because they do not exploit the redundancy in adjacent image patches. Finally, most sparse coding algorithms (Olshausen and Field, 1997; Lee et al., 2007; Murray and Kreutz-Delgado, 2006; Rozell et al., 2008; Efron et al., 2004) have found limited applications in vision due to the high computational cost of the iterative optimization required to compute the feature descriptor.

## 2.1 Predictive Sparse Decomposition

In order to alleviate these problems, we have introduced an algorithm named *"Predictive Sparse Decomposition"*, (PSD), which has the following characteristics: **1.** it produces efficient, feed-forward filter banks that include a point-wise non-linearity; **2.** the training procedure is deterministic (no sampling required, as with Restricted Boltzmann Machines); **3.** it learns to produce high-dimensional *sparse features*, which are suitable for subsequent pooling, and which enhance class discriminability. Although the filter banks are eventually applied to entire images, the PSD algorithm trains them on individual patches (or stacks of patches from multiple input feature maps) whose size is equal to the size of the filters. The starting point of PSD is the well-known sparse modeling algorithm proposed by Olshausen and Field (Olshausen and Field, 1997) which, unfortunately does not produce "direct" filters, but "reverse" filters (or dictionary elements). To alleviate the need for an optimization process for obtaining the feature representation, the PSD method (Kavukcuoglu et al.) trains a simple (feed-forward) regressor (or *encoder*) to approximate the sparse solution $z^*$ for each $x \in \mathcal{X}$ in the training set. The regressor $\mathcal{F}_e(x; K)$ takes the form of eq. (2.3) on a patch the size of the filters ($K = k, g$ collectively denotes all the filter coefficients).

$$\mathcal{F}_e(x; K) = g \cdot \tanh(k^T x) \tag{2.3}$$

where $k \in \mathcal{R}^{n \times m}$ and $g \in \mathcal{R}^n$. During training of the dictionary $\mathcal{D}$ and the predictor function parameters $K$, the optimal feature vector $z^*$ is obtained by minimizing the energy function $E_{PSD}(x, z, \mathcal{D}, K)$, defined as follows:

$$\begin{aligned} E_{PSD} &= \frac{1}{2}\|x - \mathcal{D}z\|_2^2 + \lambda\|z\|_1 + \alpha\|z - \mathcal{F}_e(x; K)\|_2^2 && (2.4) \\ z^* &= \arg\min_z \; E_{PSD}(x, z, \mathcal{D}, K) && (2.5) \end{aligned}$$

This augmented energy function, averaged over an input dataset $\mathcal{X}$, is minimized to train the the dictionary $\mathcal{D}$ and encoder ($\mathcal{F}_e$) parameters $K$. Details of the learning procedure are very similar to conventional sparse modeling algorithms and are explained in the next section.

## 2.2 Learning

As with (Olshausen and Field, 1997), learning proceeds by minimizing the loss over the training set $\mathcal{X}$:

$$\mathcal{L}_{PSD}(\mathcal{D}, K) = \frac{1}{|\mathcal{X}|}\sum_{i=1}^{|\mathcal{X}|} \min_{\mathcal{Z}^i} E_{PSD}(\mathcal{X}^i, \mathcal{Z}^i, \mathcal{D}, K) \qquad (2.6)$$

The learning procedure simultaneously optimizes $\mathcal{D}$ (dictionary) and $K$ (predictor parameters). The goal is to find the optimal value of the basis functions $\mathcal{D}$, as well as the value of the parameters in the regressor ($K$), thus minimizing $\mathcal{L}_{PSD}$ in Eqn. (2.6). Learning proceeds by a stochastic on-line coordinate gradient descent algorithm, alternating the following two steps for each training sample $x \in \mathcal{X}$:

1. **Inference:** Keeping the parameters $K$ and $\mathcal{D}$ constant, minimize $E_{PSD}$ of eq. (2.4) with respect to $z$, starting from the initial value provided by the regressor $\mathcal{F}_e(x; K)$, thus obtaining $z^*$.

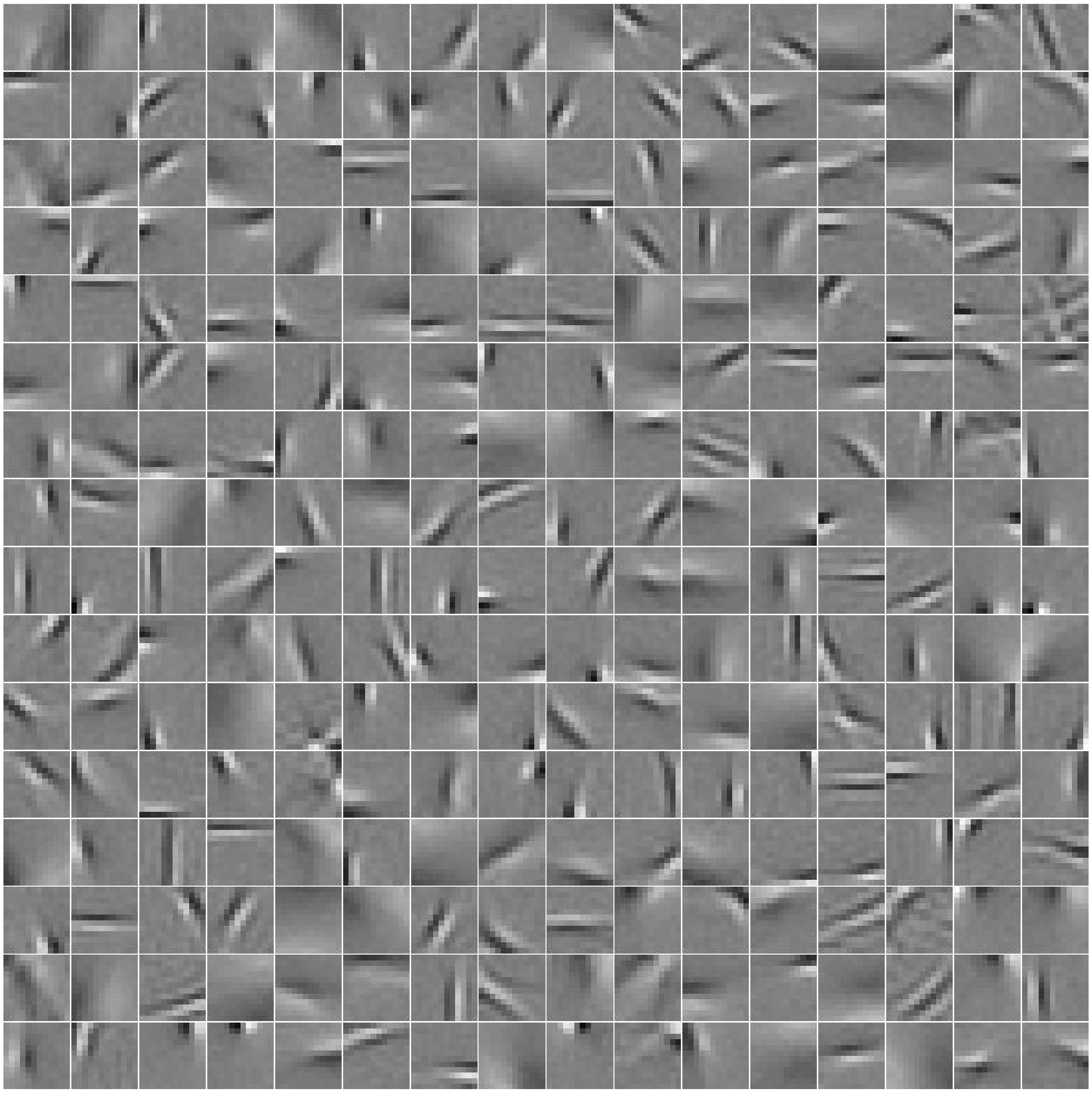

Figure 2.1: Learned dictionary elements $\mathcal{D}$ with PSD algorithm on natural image patches. Each dictionary element is rearranged to match the shape of the input patch ($12 \times 12$). Learned dictionary elements are localized oriented edge detectors at various scales, positions and orientations. The system is trained on grayscale natural image patches without any pre-processing. 0 is displayed as mid-gray level 128.

2. **Update:** Using the optimal value ($z^*$) of the coefficients $z$ provided by the previous step, update the parameters $K$ and $\mathcal{D}$ by one step of stochastic gradient descent. The update is: $U \leftarrow U - \eta \frac{\partial \mathcal{L}_{PSD}}{\partial U}$, where $U$ collectively denotes $\{K, \mathcal{D}\}$ and $\eta$ is the step size. The columns of $\mathcal{D}$ are then re-scaled to unit norm.

We set $\alpha = 1$ for all experiments. We found that training the set of basis functions $\mathcal{D}$ first, then subsequently training the regressor, yields similar performance in terms of recognition accuracy. However, when the regressor is trained afterwards, the overall training time is considerably longer. Learned filters using the PSD algorithm are shown in Fig. 2.1.

Interestingly, we recover different algorithms depending on the value of the parameter $\alpha$:

- $\alpha = 0$. The loss of eq. (2.6) reduces to the one in eq. (2.2). The learning algorithm becomes similar to Olshausen and Field's sparse coding algorithm (Olshausen and Field, 1997). The regressor can then be trained *separately* from the set of basis functions $\mathcal{D}$.

- $\alpha \in (0, +\infty)$. The parameters are updated taking into account also the constraint on the representation, using the same principle employed by SESM training (Ranzato et al., 2007a), for instance.

- $\alpha \rightarrow +\infty$. The additional constraint on the representation (the third term in eq. (2.6)) becomes an equality, i.e. $Z = \mathcal{F}_e(x; K)$, and the model becomes similar to an auto-encoder neural network with a sparsity regularization term acting on the internal representation $z$ instead of a regularization acting on the parameters $K$ and $\mathcal{D}$. This is equivalent to minimizing the following energy function:

$$\frac{1}{2}\|x - \mathcal{D}\mathcal{F}_e(x;K)\|_2^2 + \lambda\|\mathcal{F}_e(x;K)\|_1 \tag{2.7}$$

Once the parameters are learned, inferring the representation $z$ can be done in two ways.

1. **Optimal inference** is defined as obtaining optimal representation $z^*$ by solving for eq. (2.5). We perform this step by running an iterative gradient descent optimization algorithm involving two possibly large matrix-vector multiplications at each iteration (one for computing the value of the objective, and one for computing the derivatives through $\mathcal{D}$).

2. **Approximate inference**, on the other hand sets the representation ($z$) to the value produced by $\mathcal{F}_e(x;K)$ as given in eq. (2.3), involving only a forward propagation through the regressor, i.e. a single matrix-vector multiplication.

It can be seen that the optimal inference procedure is significantly more costly compared to the approximate inference procedure. For recognition purposes, we only use approximate inference.

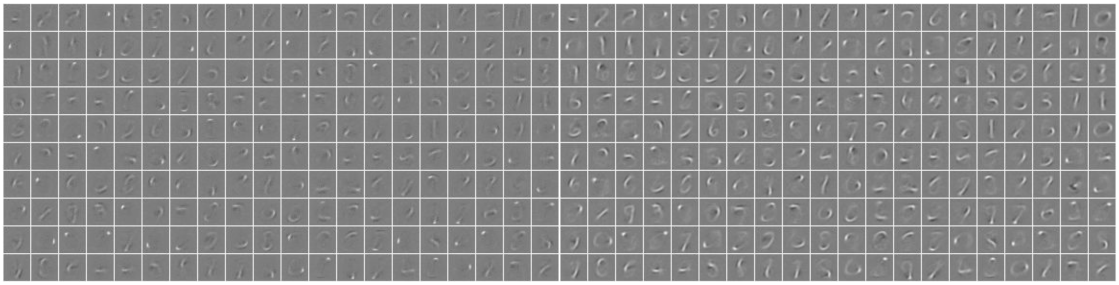

Figure 2.2: Learned encoder filters $k$ (left) and dictionary elements $\mathcal{D}$ (right) obtained using PSD algorithm on the MNIST dataset. Each dictionary element is rearranged to match the shape of input patch ($28 \times 28$). Learned dictionary elements are strokes that make up a whole digit. The system is trained on MNIST training set without any pre-processing. 0 is displayed as mid-gray level 128.

## 2.3 Experimental Evaluation

First, we demonstrate that the proposed algorithm (PSD) is able to produce good features for recognition by comparing to other unsupervised feature extraction algorithms, Principal Components Analysis (PCA), Restricted Boltzman Machine (RBM), and Sparse Encoding Symmetric Machine (SESM). Then, we compare the recognition accuracy and inference time of PSD feed-forward approximation to feature sign algorithm (Lee et al., 2007), on the Caltech 101 dataset (Fei-Fei et al., 2004).

We calculate a 256 dimensional representation for each $28 \times 28$ digit patch using PSD, RBM, PCA, and SESM and compare the classification error rate using a regularized linear logistic classifier trained on the corresponding representations.

We have used the first 20000 training samples for unsupervised training of different methods and a variable number of labelled samples for training the classifier. Fig. 2.2 shows encoder filters $k$ and dictionary $\mathcal{D}$. The system learns localized strokes detectors so that several of these can be combined linearly to reconstruct any given digit. Fig. 2.3 shows the classification error comparison between different methods as a function of the corresponding reconstruction error. It can be seen that, even though PSD provides the **worst reconstruction error**, it can achieve the **best recognition accuracy** on the test set under different number of training samples per class.

### 2.3.1 Comparison with Exact Algorithms

In order to quantify how well our jointly trained predictor given in eq. (2.3) approximates the optimal representations obtained by minimizing the loss in eq. (2.6) and the optimal representations that are produced by an exact algorithm minimizing eq. (2.2) such as feature sign (FS) (Lee et al., 2007), we measure the average signal to noise ratio[1] (SNR) over a test dataset of 20,000 natural image patches of size 9x9. The data set of images was constructed by randomly picking 9x9 patches from the images of the Berkeley dataset converted to gray-scale values, and these patches were normalized to have zero mean and unit standard deviation. The algorithms were trained to learn sparse codes with 64 units[2].

[1] $SNR = 10log_{10}(\sigma^2_{signal}/\sigma^2_{noise})$

[2] Principal Component Analysis shows that the effective dimensionality of 9x9 natural image patches is about 47 since the first 47 principal components capture 95% of the variance in the

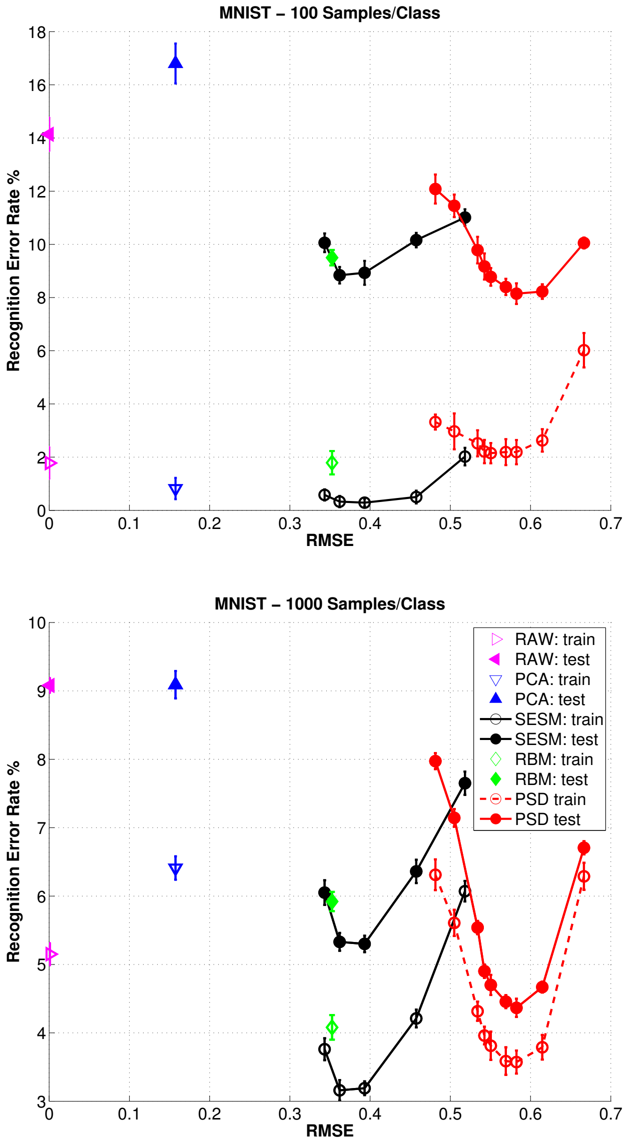

Figure 2.3: Classification error on MNIST as a function of the reconstruction error using raw pixel values and, PCA, RBM, SESM and PSD features. Left-to-Right : 100-1000 samples per class are used for training a linear classifier on the features. The unsupervised algorithms were trained on the first 20,000 training samples of the MNIST dataset (LeCun and Cortes, 1998).

| Comparison (Signal / Approximation) | Signal to Noise Ratio (SNR) |
|---|---|
| 1. PSD Optimal / PSD Predictor | 8.6 |
| 2. FS / PSD Optimal | 5.2 |
| 3. FS / PSD Predictor | 3.1 |
| 4. FS / Regressor | 3.2 |

Table 2.1: Comparison between representations produced by FS (Lee et al., 2007) and PSD. In order to compute the SNR, the noise is defined as $(Signal - Approximation)$.

We compare representations obtained by "PSD Predictor" using the *approximate* inference, "PSD Optimal" using the *optimal* inference, "FS" minimizing eq. (2.2) with (Lee et al., 2007), and "Regressor" that is separately trained to approximate the exact optimal codes produced by FS. The results given in table 2.1 show that PSD direct predictor achieves about the same SNR on the true optimal sparse representations produced by FS, as the Regressor that was trained to predict these representations.

Despite the lack of absolute precision in predicting the exact optimal sparse codes, PSD predictor achieves even slightly better performance in recognition.

data. Hence, a 64-dimensional feature vector is actually an overcomplete representation for these 9x9 image patches.

### 2.3.2 Recognition Performance

We test the performance of using PSD filters in a large-scale object recognition task using Caltech 101 (Fei-Fei et al., 2004) dataset. The Caltech 101 dataset is pre-processed in the following way similar to (Pinto et al., 2008):

**1)** Each image is converted to gray-scale,

**2)** Down-sampled so that the longest side is 151 pixels,

**3)** The image is globally normalized by subtracting the image mean and dividing by the image standard deviation,

**4)** The image ($x$) is locally contrast normalized by subtracting the weighted local mean ($\mu$) from each pixel and dividing each pixel by the weighted local standard deviation ($\sigma$), only if $\sigma$ is larger than ($c = 1$) (eq. 2.8-2.11). The weighting $w$ is a 9x9 Gaussian window with standard deviation 1.591.

$$\mu_{jk} = \sum_{ipq} w_{pq}.x_{i,j+p,k+q} \tag{2.8}$$

$$v_{ijk} = x_{ijk} - \mu_{jk} \tag{2.9}$$

$$\sigma_{jk} = \sqrt{\sum_{ipq} w_{pq}.v^2_{i,j+p,k+q}} \tag{2.10}$$

$$y_{ijk} = v_{ijk}/\max(c, \sigma_{jk}) \tag{2.11}$$

where $i$ is the feature map index, $j, k$ are the pixel location indices.

**5)** The image is 0-padded to 143x143 pixels.

64 feature detectors (either produced by FS or PSD predictor) were plugged into

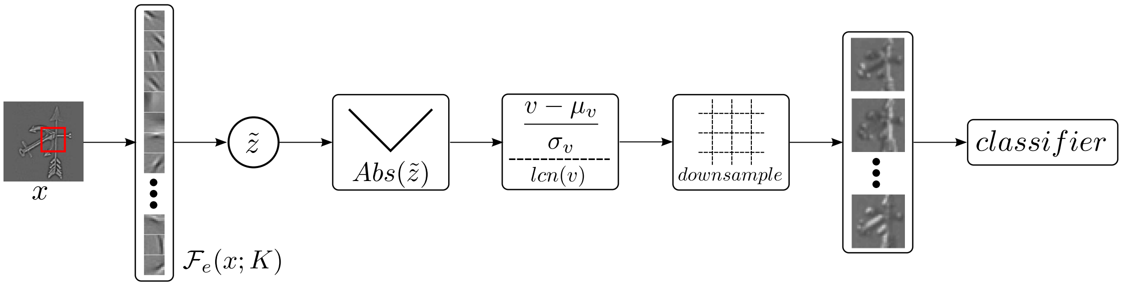

Figure 2.4: Object recognition architecture: linear adaptive filter bank, followed by *abs* rectification, average down-sampling and linear SVM classifier.

an image classification system that:

**1)** Used the sparse coding algorithms convolutionally to produce 64 feature maps of size 128x128 for each pre-processed image,

**2)** Applied an absolute value rectification,

**3)** Applied local contrast normalization as explained in eq. 2.8-2.11, this time the local neighborhood is again $9 \times 9$ but over all 64 features.

**4)** Computed an average down-sampling to a spatial resolution of 30x30 and

**5)** Used a linear SVM classifier to recognize the object in the image (see fig. 2.4).

Using this system with 30 training images per class we can achieve 53% accuracy on the Caltech 101 dataset.

Since FS finds exact sparse codes, its representations are generally sparser than those found by PSD predictor trained with the same value of sparsity penalty $\lambda$. Hence, we compare the recognition accuracy against the *measured* sparsity level of the representation as shown in fig. 2.5(b). PSD is not only able to achieve better accuracy than exact sparse coding algorithms, but also, it does it much

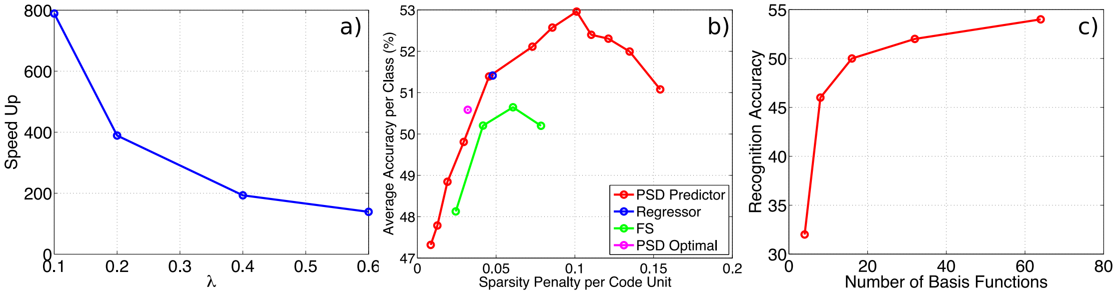

Figure 2.5: **a)** Speed up for inferring the sparse representation achieved by PSD predictor over FS for a code with 64 units. The feed-forward extraction is more than 100 times faster. **b)** Recognition accuracy versus measured sparsity (average $\ell^1$ norm of the representation) of PSD predictor compared to the representation of the FS algorithm. A difference within 1% is not statistically significant. **c)** Recognition accuracy as a function of the number of basis functions.

more efficiently. Fig. 2.5(a) demonstrates that our feed-forward predictor extracts features more than 100 times faster than feature sign. In fact, the speed up is over 800 when the sparsity is set to the value that gives the highest accuracy shown in fig. 2.5(b).

Finally, we observe that these sparse coding algorithms are somewhat inefficient when applied convolutionally. Many feature detectors are the translated versions of each other as shown in fig. 2.4(a). Hence, the resulting feature maps are highly redundant. This might explain why the recognition accuracy tends to saturate when the number of filters is increased as shown in fig. 2.5(c).

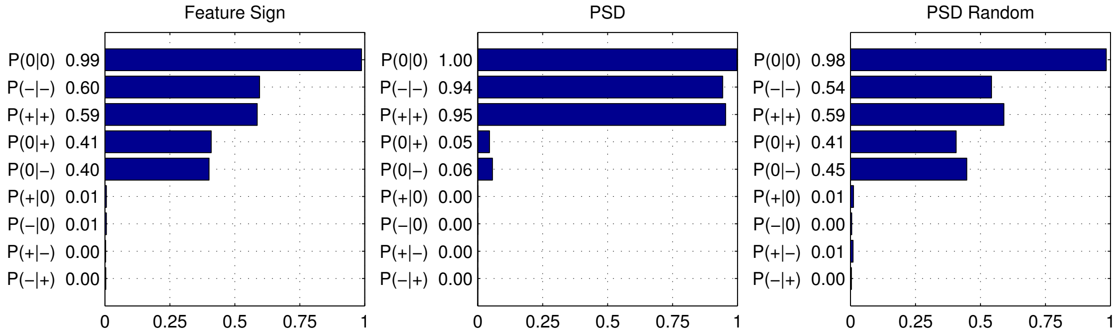

Figure 2.6: Conditional probabilities for sign transitions between two consecutive frames. For instance, $P(-|+)$ shows the conditional probability of a unit being negative given that it was positive in the previous frame. The figure on the right is used as baseline, showing the conditional probabilities computed on pairs of *random* frames.

### 2.3.3 Stability

In order to quantify the stability of PSD and FS, we investigate their behavior under naturally changing input signals. For this purpose, we train a basis set with 128 elements, each of size 9x9, using the PSD algorithm on the Berkeley (Martin et al., 2001) dataset. This basis set is then used with FS on the standard "foreman" test video together with the PSD Predictor. We extract 784 uniformly distributed patches from each frame with a total of 400 frames.

For each patch, a 128 dimensional representation is calculated using both FS and the PSD predictor. The stability is measured by the number of times a unit of the representation changes its sign, either negative, zero or positive, between two consecutive frames. Since the PSD predictor does not generate exact zero values, we threshold its output units in such a way that the average number of zero units equals the one produced by FS (roughly, only the 4% of the units are non-zero). The transition probabilities are given in Figure 2.6. It can be seen from this figure that the PSD predictor generates a more stable representation of slowly varying natural frames compared to the representation produced by the exact optimization algorithm.

# Chapter 3

# Modeling Invariant Representations

As discussed before, most feature extraction systems contain a pooling stage for encoding invariance to small translations, and in particular cases (Lowe, 2004), invariance to rotations can also be hard coded into the architecture of the feature extractor. Our aim is to learn the filter bank stage and the pooling stage simultaneously, in such a way that the filters in the same pool extract similar features. Rather than learning descriptors that are merely invariant to small shift (a property that can easily be built by hand), our goal is to learn descriptors that are also invariant to other types of transformations, such as rotations and certain distortions. Our solution is to pre-wire (before learning) which filters' outputs are pooled together, and to let the underlying filters learn their coefficients by minimizing the sparsity criterion on the pooling units. As a result, filters that are pooled together end up extracting similar features. A similar approach was also proposed in (Hyvärinen and Hoyer, 2001) for the special case of square ICA formulation. (Yuan and Lin, 2006) also introduced group lasso formulation for sparse coding. However, in this case each group is unique, there is no sharing of code components between groups. In this work, we extend these model to overcomplete sparse modeling case with overlapping group definitions so that one code component can be a member of more than one pooling group and additionally train a predictor function to approximate pooled coefficients in a feed-forward manner.

In a recent work (Ranzato et al., 2007b) proposed an architecture that encodes shift invariance in the pooling stage and the filterbank is learned. On the other hand, several authors have proposed methods to learn pooled features in the context of computational models of the mammalian primary visual cortex. These models rely on imposing some sort of sparsification criteria on small groups of filter outputs (Kohonen, 1996; Hyvarinen and Hoyer, 2000 Jul; Osindero et al., 2006). When the filters that are pooled together are organized in a regular array (1D or 2D), the filters form *topographic maps* in which nearby filters extract similar features.

In regular sparse coding, the sparsity penalty has a simple form such as $\ell^1$ as shown below. In this form, each component of the representation $z$ is penalized individually proportional to absolute value of its activation. This property enforces competition between dictionary elements for representing the input, but at the same time introduces instability since similar inputs can be represented by different dictionary elements.

$$\min \frac{1}{2}||x - \mathcal{D}z||_2^2 + \lambda|z|_1 \tag{3.1}$$

In order to overcome this problem, we have introduced an extension to the PSD algorithm, named *"Invariant Predictive Sparse Decomposition"* (IPSD), in our work *"Learning invariant features through topographic filter maps"* (Kavukcuoglu et al., 2009).

We first arrange the $z$'s into a 2D map (or some other topology) and then pool the squared coefficients of $z$ across overlapping windows. Then, the square of the the filter outputs within each sub-window are summed, and their square roots are computed. More formally, let the map of $z$ contain several overlapping neighborhoods $P_i$. Within each neighborhood $i$, we sum the squared coefficients $z_j$ (weighted by a fixed Gaussian weighting function centered in the neighborhood that sum to 1) and then take the square root. This gives the activation $v_i = \sqrt{\sum_{j \in P_i} w_j z_j^2}$, where $w_j$ are the Gaussian weights. The overall sparsity penalty is the sum of each neighborhood's activation: $\sum_{i=1}^{|P|} v_i$. Figure 3.1(a) illustrates this scheme. Thus, the overall objective function is now:

$$E = \frac{1}{2}||x - \mathcal{D}z||_2^2 + \lambda \sum_{i=1}^{|P|} \sqrt{\sum_{j \in P_i} w_j z_j^2} \tag{3.2}$$

Note that the pooling neighborhoods $P_i$ in eq. (3.2) overlap with others which is not the case is group lasso formulation (Yuan and Lin, 2006). The modified sparsity

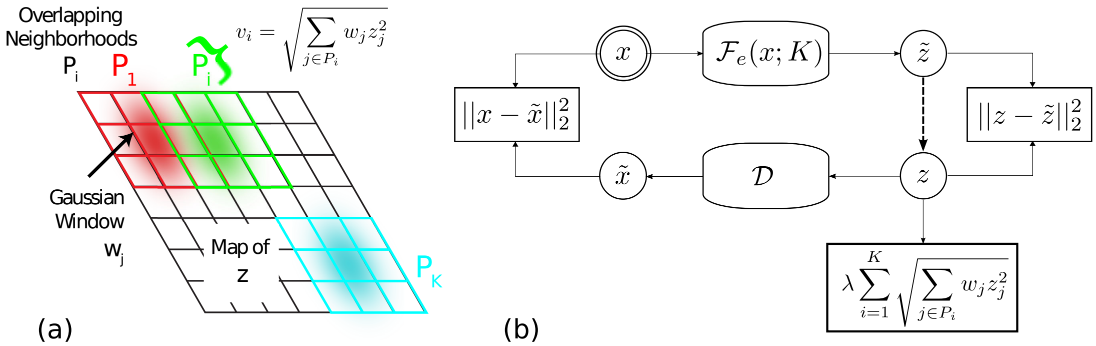

Figure 3.1: **(a)**: The structure of the block-sparsity term which encourages the basis functions in $\mathcal{D}$ to form a topographic map. See text for details. **(b)**: Overall architecture of the loss function, as defined in eq. (3.4). In the generative model, we seek a feature vector $z$ that simultaneously approximates the input $x$ via a dictionary of basis functions $\mathcal{D}$ and also minimizes a sparsity term. Since performing the inference at run-time is slow, we train a prediction function $\mathcal{F}_e(x;K)$ (dashed lines) that directly predicts the optimal $z$ from the input $x$. At run-time we use only the prediction function to quickly compute $z$ from $x$, from which the invariant features $v_i$ can computed.

term has a number of subtle effects on the nature of $z$ that are not immediately obvious:

- The square root in the sum over $i$ encourages sparse activations *across* neighborhoods since a few large activations will have lower overall cost than many small ones.
- *Within* each neighborhood $i$, the coefficients $z_j$ are encouraged to be similar to one another due to the $z_j^2$ term (which prefers many small coefficients to a few large ones). This has the effect of encouraging similar basis functions in $\mathcal{D}$ to be spatially close in the 2D map.
- As the neighborhoods overlap, these basis functions will smoothly vary across the map, so that the coefficients $z_j$ in any given neighborhood $i$ will be similar.
- If the size of the pooling regions is reduced to a single $z$ element, then the sparsity term is equivalent to that of eq. (2.2).

The modified sparsity term means that by minimizing the loss function $\mathcal{L}_I$ in eq. (3.2) with respect to both the coefficients $z$ and the dictionary $\mathcal{D}$, the system learns a set of basis functions in $\mathcal{D}$ that are laid out in a *topographic map* on the 2D grid.

Since the nearby coefficients $z_j$ will be similar for a given input $x$. This also means that if this input is perturbed slightly then the pooled response within a given neighborhood will be minimally affected, since a decrease in the response of one filter will be offset by an increased response in a nearby one. Hence, we can obtain a locally robust representation by taking the pooled activations $v_i$ as features, rather than $z$ as is traditionally done.

Since invariant representations encode similar patterns with the same representation, they can be made more compact. Put another way, this means that the dimensionality of $v$ can be made lower than the dimensionality of $z$ without loss of useful information. This has the triple benefit of requiring less computation to extract the features from an image, requiring less memory to store them, and requiring less computation to exploit them.

The 2D map over $z$ uses circular boundary conditions to ensure that the pooling wraps smoothly around at the edges of the map.

We can introduce a predictor function as given in eq. (2.3) and the overall invariant energy function becomes:

$$\begin{aligned} E_{IPSD} &= \frac{1}{2}\|x - \mathcal{D}z\|_2^2 + \lambda \sum_{i=1}^{|P|} \sqrt{\sum_{j \in P_i} w_j z_j^2} + \alpha \|z - \mathcal{F}_e(x;K)\|_2^2 && (3.3) \\ z^* &= \arg\min_z E_{IPSD}(x, z, \mathcal{D}, K) && (3.4) \end{aligned}$$

By minimizing this energy function, a predictor ($\mathcal{F}_e(x;K)$) can be trained to produces an approximation to the optimal representation $z^*$. Learning proceeds in exactly the same way as for PSD by minimizing the energy function over the training set $\mathcal{X}$:

$$\mathcal{L}_{IPSD}(\mathcal{D}, K) = \frac{1}{|\mathcal{X}|} \sum_{i=1}^{|\mathcal{X}|} \min_{\mathcal{Z}^i} E_{IPSD}(\mathcal{X}^i, \mathcal{Z}^i, \mathcal{D}, K) \qquad (3.5)$$

to train the dictionary $\mathcal{D}$ and predictor parameters $K$. Once the parameters are learned, computing the invariant representation $v$ can be performed by a simple

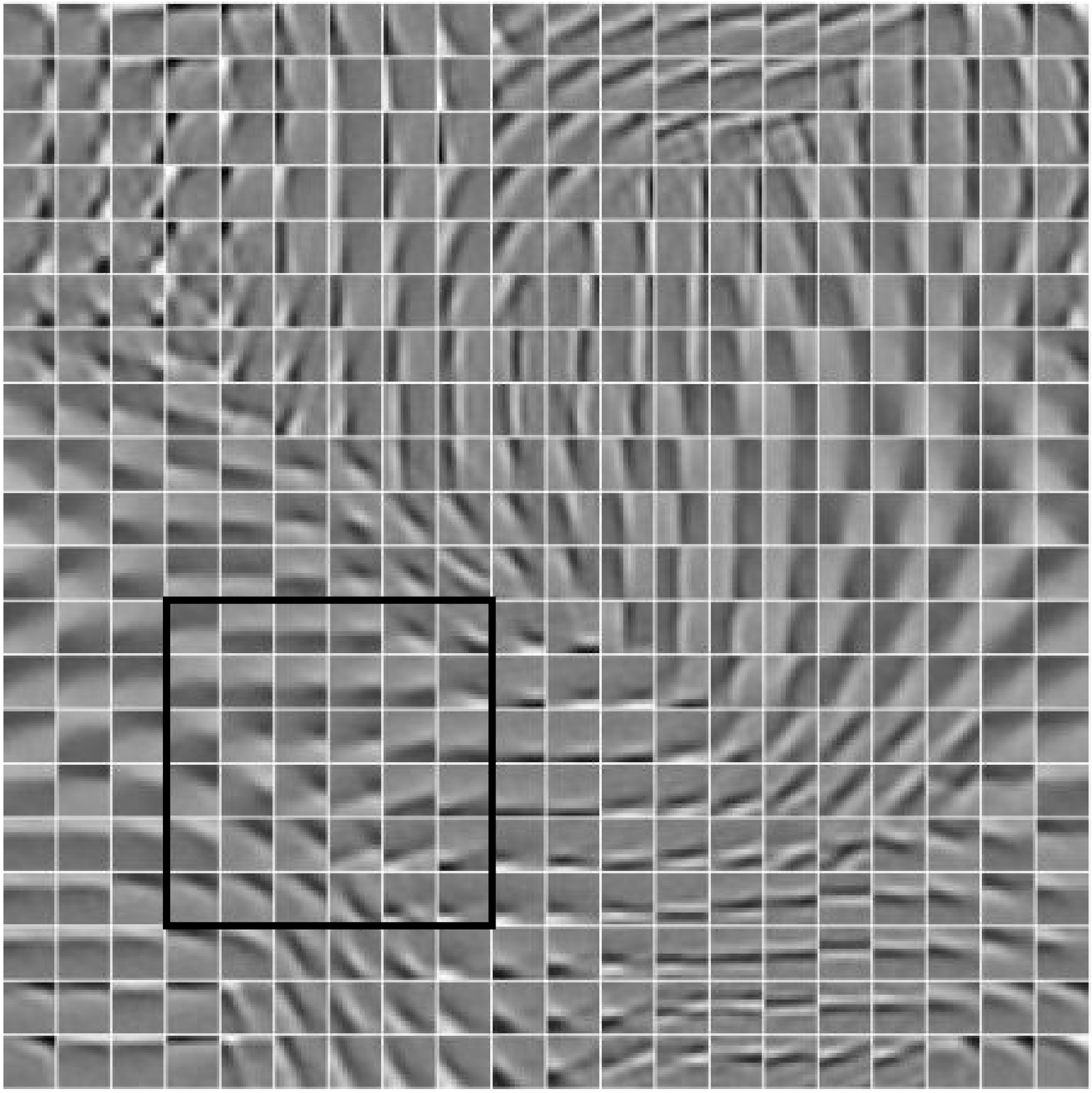

Figure 3.2: Topographic map of feature detectors learned from natural image patches of size 12x12 pixels by optimizing $\mathcal{L}_{IPSD}$ in eq. (3.5). There are 400 filters that are organized in 6x6 neighborhoods. Adjacent neighborhoods overlap by 4 pixels both horizontally and vertically. Notice the smooth variation within a given neighborhood and also the circular boundary conditions.

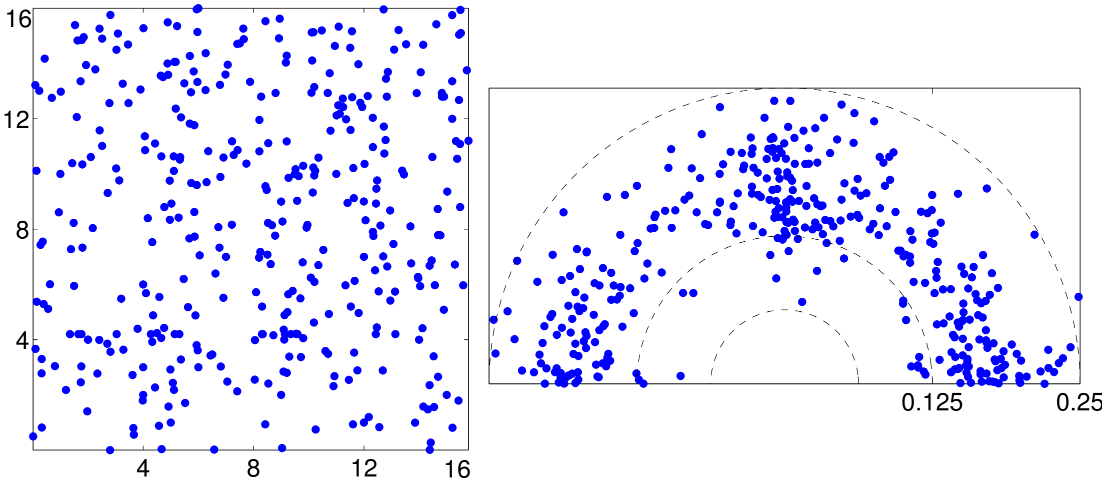

Figure 3.3: Analysis of learned filters by fitting Gabor functions, each dot corresponding to a feature. Left: Center location of fitted Gabor. Right: Polar map showing the joint distribution of orientation (azimuthally) and frequency (radially) of Gabor fit.

feed-forward propagation through the regressor $\mathcal{F}_e(x; K)$, and then propagating $z$ into $v$ through $v_i = \sqrt{\sum_{j \in P_i} w_j z_j^2}$.

Fig. 3.2 shows a typical topographic map learned by the proposed method from natural image patches. Each tile shows a filter in $\mathcal{D}$ corresponding to a particular $z_i$. In the example shown, the input images are patches of size $12 \times 12$ pixels, and there are 400 basis functions, hence, 400 units $z_i$ arranged in a $20 \times 20$ lattice. The neighborhoods over which the squared activities of $z_i$'s are pooled are $6 \times 6$ windows, and they overlap by 4 in both the vertical and the horizontal direction.

By varying the way in which the neighborhoods are pooled, we can change the properties of the map. Larger neighborhoods make the filters in each pool increasingly similar. A larger overlap between windows makes the filters vary more smoothly across different pools. A large sparsity value $\lambda$ makes the feature detectors learn less localized patterns that look like those produced by k-means clustering because the input has to be reconstructed using a small number of basis functions. On the other hand, a small sparsity value makes the feature detectors look like non-localized random filters because any random overcomplete basis set can produce good reconstructions (effectively, the first term in the loss of eq.( 3.4) dominates).

The map in Fig. 3.2 has been produced with an intermediate sparsity level of $\lambda = 3$. The chosen parameter setting induces the learning algorithm to produce a smooth map with mostly localized edge detectors in different positions, orientations, and scales. These feature detectors are nicely organized in such a way that neighboring units encode similar patterns. A unit $v_i$, that connects to the sum of the squares of units $z_j$ in a pool is invariant because these units represent similar features, and small distortions applied to the input, while slightly changing the $z_j$'s within a pool, are likely to leave the corresponding $v_i$ unaffected.

While the sparsity level, the size of the pooling windows and their overlap should be set by cross-validation, in practice we found that their exact values do not significantly affect the kind of features learned. In other words, the algorithm is quite robust to the choice of these parameters, probably because of the many constraints enforced during learning.

In the following section, before exploring the properties of the invariant features obtained, we first study the topographic map produced by our training scheme. First, we make an empirical evaluation of the invariance achieved by these representations under translations and rotations of the input image. Second, we assess the discriminative power of these invariant representations on recognition tasks in three different domains: (i) generic object categories using the Caltech 101 dataset; (ii) generic object categories from a dataset of very low resolution images and (iii) classification of handwriting digits using the MNIST dataset. In these experiments we compare IPSD's learned representations with the SIFT descriptor (Lowe, 2004) that is considered a state-of-the-art descriptor in computer vision.

## 3.1 Analyzing Invariance to Transformations

In this experiment we study the invariance properties of the learned representation under simple transformations. We have generated a dataset of 16x16 natural image patches under different translations and rotations. Each patch is presented to the predictor function that produces a 128 dimensional descriptor (chosen to be the same size as SIFT) composed of $v$'s. A representation can be considered locally invariant if it does not change significantly under small transformations of the input. Indeed, this is what we observe in Fig. 3.4. We compare the mean squared difference between the descriptor of the reference patch and the descriptor of the transformed version, averaged over many different image patches. The figure compares proposed descriptor against SIFT with a varying horizontal shift for 0

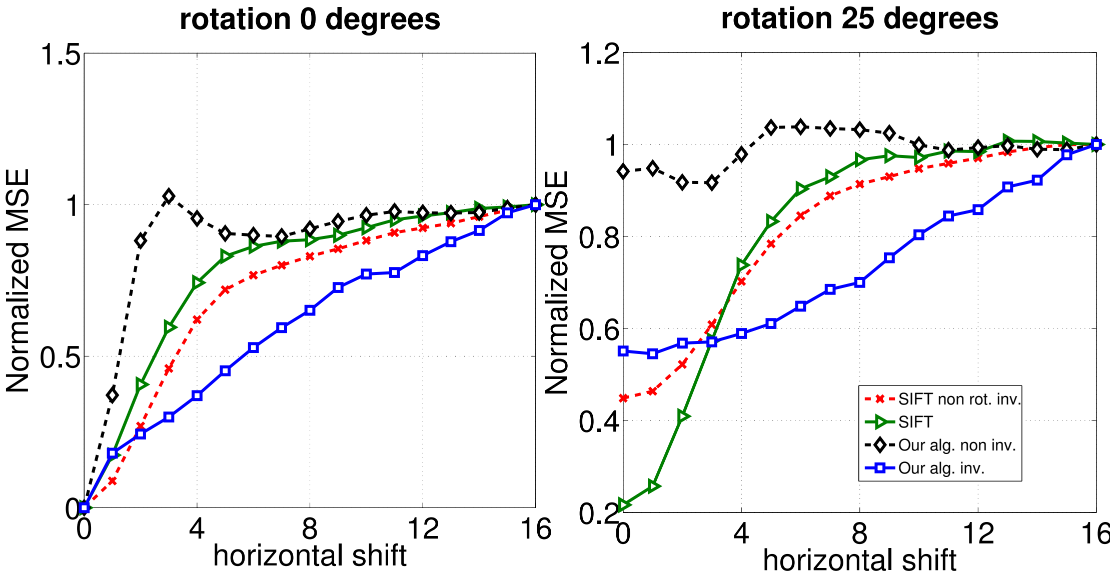

Figure 3.4: Mean squared error (MSE) between the representation of a patch and its transformed version. On the left panel, the transformed patch is horizontally shifted. On the right panel, the transformed patch is first rotated by 25 degrees and then horizontally shifted. The curves are an average over 100 patches randomly picked from natural images. Since the patches are 16x16 pixels in size, a shift of 16 pixels generates a transformed patch that is quite uncorrelated to the original patch. Hence, the MSE has been normalized so that the MSE at 16 pixels is the same for all methods. This allows us to directly compare different feature extraction algorithms: non-orientation invariant SIFT, SIFT, IPSD trained to produce non-invariant representations (i.e. pools have size 1x1 and the algorithm becomes equivalent to PSD), and IPSDtrained to produce invariant representations. All algorithms produce a feature vector with 128 dimensions. IPSD produces representations that are more invariant to transformations than the other approaches.

and 25 degrees initial rotation. Very similar results are found for vertical shifts and other rotation angles.

On the left panel, we can see that the mean squared error (MSE) between the representation of the original patch and its transformation increases linearly as we increase the horizontal shift. The MSE of IPSD representation is generally lower than the MSE produced by features that are computed using SIFT, a non-rotation invariant version of SIFT, and a non-invariant representation produced by the proposed method (that was trained with pools of size 1x1) (Kavukcuoglu et al.). A similar behavior is found when the patch is not only shifted, but also rotated. When the shift is small, SIFT has lower MSE, but as soon as the translation becomes large enough that the input pattern falls in a different internal sub-window, the MSE increases considerably. Instead learned representations using IPSD seem to be quite robust to shifts, with an overall lower area under the curve. Note also that traditional sparse coding algorithms are prone to produce unstable representations under small distortions of the input. Because each input has to be encoded with a small number of basis functions, and because the basis functions are highly tuned in orientation and location, a small change in the input can produce drastic changes in the representation. This problem is partly alleviated by our approximate inference procedure that uses a smooth predictor function. However, this experiment shows that this representations is fairly unstable under small distortions, when compared to the invariant representations produced by IPSD and SIFT.

## 3.2 Generic Object Recognition

The invariance properties of the IPSD features are even better than the SIFT features, however more important is the recognition performance using these features. For that purpose we have used Caltech-101 (Fei-Fei et al., 2004) dataset.

The Caltech-101 dataset is pre-processed in the same way as previous experiments, by resizing to 151×151 gray scale images followed by contrast normalization as explained in section 2.3.2. We have then trained IPSD;on 50,000 16x16 patches randomly extracted from the pre-processed images. The topographic map used has size 32x16, with the pooling neighborhoods being 6x6 and an overlap of 4 coefficients between neighborhoods. Hence, there are a total of 512 units that are used in 128 pools to produce a 128-dimensional representation that can be compared to SIFT. After training IPSD in an unsupervised way, we use the predictor function to infer the representation of one whole image by: (i) running the predictor function on 16x16 patches spaced by 4 pixels to produce 128 maps of features of size 32x32; (ii) the feature maps are locally normalized (neighborhood of 5x5) and low-pass filtered with a boxcar filter (5x5) to avoid aliasing; (iii) the maps are then projected along the leading 3060 principal components (equal to the number of training samples), and (iv) a supervised linear SVM[1] is trained to recognize the object in each corresponding image. The overall scheme is shown in Fig. 3.5.

Table 4.1 reports the recognition results for this experiment. With a linear classifier similar to (Pinto et al., 2008), IPSD features outperform SIFT and the

[1] We have used LIBSVM package available at http://www.csie.ntu.edu.tw/ cjlin/libsvm

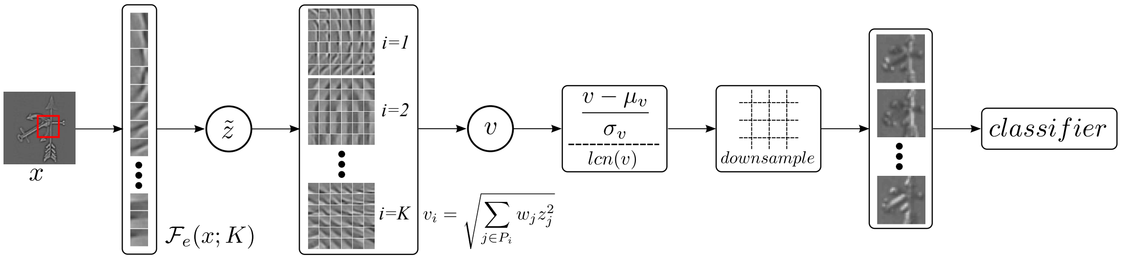

Figure 3.5: Diagram of the recognition system, which is composed of an invariant feature extractor that has been trained unsupervised, followed by a supervised linear SVM classifier. The feature extractor process the input image through a set of filter banks, where the filters are organized in a two dimensional topographic map. The map defines pools of similar feature detectors whose activations are first non-linearly transformed by a hyperbolic tangent non-linearity, and then, multiplied by a gain. Invariant representations are found by taking the square root of the sum of the squares of those units that belong to the same pool. The output of the feature extractor is a set of feature maps that can be fed as input to the classifier. The filter banks and the set of gains is learned by the algorithm. Recognition is very fast, because it consists of a direct forward propagation through the system.

| **Method** | **Av. Accuracy/Class** (%) |
|---|---|
| **local norm$_{\mathbf{5\times5}}$ + boxcar$_{\mathbf{5\times5}}$ + PCA$_{\mathbf{3060}}$ + linear SVM** | |
| IPSD (24x24) | **50.9** |
| SIFT (24x24) (non rot. inv.) | **51.2** |
| SIFT (24x24) (rot. inv.) | **45.2** |
| **local norm$_{\mathbf{9\times9}}$ + Spatial Pyramid Match Kernel SVM** | |
| SIFT (Lazebnik et al., 2006) | **64.6** |
| IPSD (34x34) | **59.6** |
| IPSD (56x56) | **62.6** |
| IPSD (120x120) | **65.5** |

Table 3.1: Recognition accuracy on Caltech 101 dataset using a variety of different feature representations and two different classifiers. The PCA + linear SVM classifier is similar to (Pinto et al., 2008), while the Spatial Pyramid Matching Kernel SVM classifier is that of (Lazebnik et al., 2006). IPSD is used to extract features with three different sampling step sizes over an input image to produce 34x34, 56x56 and 120x120 feature maps, where each feature is 128 dimensional to be comparable to SIFT. Local normalization is **not** applied on SIFT features when used with Spatial Pyramid Match Kernel SVM.

model proposed by Serre and Poggio (Serre et al., 2005). However, if rotation invariance is removed from SIFT its performance becomes comparable to IPSD.

We have also experimented with the more sophisticated Spatial Pyramid Matching (SPM) Kernel SVM classifier of Lazebnik et al. (Lazebnik et al., 2006). In this experiment, we again used the same IPSD architecture on 16x16 patches spaced by 3 pixels to produce 42x42x128 dimensional feature maps, followed by local normalization over a 9x9 neighborhood, yielding 128 dimensional features over a uniform 34x34 grid. Using SPM, IPSD features achieve 59.6% average accuracy per class. By decreasing the stepping stride to 1 pixel, thereby producing 120x120 feature maps, IPSD features achieve 65.5% accuracy as shown in Table 4.1. This is comparable to Lazebnik's 64.6% accuracy on Caltech-101 (without background class) (Lazebnik et al., 2006). For comparison, our re-implementation of Lazebnik's SIFT feature extractor, stepped by 4 pixels to produce 34x34 maps, yielded 65% average recognition rate.

Note that the computation time for each descriptor is a linear function of the number of features, thus this time can be further reduced if the number of features is reduced. Fig. 3.6 shows how the recognition performance varies as the number of features is decreased.

### 3.2.1 Tiny Images classification

IPSD was compared to SIFT on another recognition task using the tiny images dataset (Torralba et al., 2008). This dataset was chosen as its extreme low-

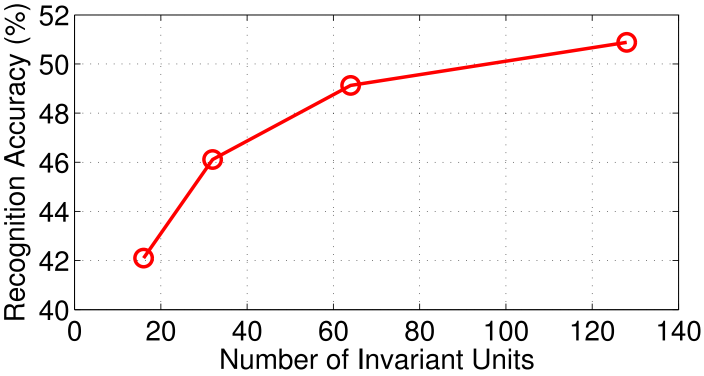

Figure 3.6: The figure shows the recognition accuracy on Caltech 101 dataset as a function of the number of invariant units. Note that the performance improvement between 64 and 128 units is below 2%, suggesting that for certain applications the more compact descriptor might be preferable.

Figure 3.7: Examples from the tiny images. Each image is 32 × 32 pixels and we use grayscale images in our experiments.

| **Performance on Tiny Images Dataset** | |
|---|---|
| **Method** | **Accuracy** (%) |
| IPSD (5x5) | **54** |
| SIFT (5x5) (non rot. inv.) | **53** |

Table 3.2: Results of recognition error rate on Tiny Images dataset. A 128 dimensional feature vector is obtained using either IPSD or SIFT over a regularly spaced 5x5 grid and afterwards a linear SVM is used for classification.

resolution provides a different setting to the Caltech 101 images. For simplicity, we selected 5 animal nouns (abyssinian cat, angel shark, apatura iris (a type of butterfly), bilby (a type of marsupial), may beetle) and manually labeled 200 examples of each. 160 images of each class were used for training, with the remaining 40 held out for testing. All images are converted to grayscale. Both IPSD with 128 pooled units and SIFT were used to extract features over 16x16 regions, spaced every 4 pixels over the 32x32 images. The resulting 5 by 5 by 128 dimensional feature maps are used with a linear SVM. IPSD features achieve 54% and SIFT features achieve a comparable 53%.

### 3.2.2 Handwriting Recognition

We use a very similar architecture to that used in the experiments above to train on the handwritten digits of the MNIST dataset (LeCun and Cortes, 1998). This is a dataset of quasi-binary handwritten digits with 60,000 images in the

| **Performance on MNIST Dataset** | |
|---|---|
| **Method** | **Error Rate** (%) |
| IPSD (5x5) | **1.0** |
| SIFT (5x5) (non rot. inv.) | **1.5** |

Table 3.3: Results of recognition error rate on MNIST dataset. A 128 dimensional feature vector is obtained using either IPSD or SIFT over a regularly spaced 5x5 grid and afterwards a linear SVM is used for classification. For comparison purposes it is worth mentioning that a Gaussian SVM trained on MNIST images without any preprocessing achieves 1.4% error rate.

training set, and 10,000 images in the test set. The algorithm was trained using 16x16 windows extracted from the original 28x28 pixel images. For recognition, 128-dimensional feature vectors were extracted at 25 locations regularly spaced over a 5x5 grid. A linear SVM trained on these features yields an error rate of **1.0%**. When 25 SIFT feature vectors are used instead of IPSD features, the error rate increases to **1.5%**. This demonstrates that, while SIFT seems well suited to natural images, IPSD produces features that can adept to the task at hand. In a similar experiment, a *single* 128-dimensional feature vector was extracted using IPSD and SIFT, and fed to a linear SVM. The error rate was 5.6% for PSD, and 6.4% for SIFT.

# Chapter 4

# Multi-Stage Architectures for Object Recognition

The PSD algorithm can be used to build multi-layer feed-forward object recognition architectures that can be trained in a layer-by-layer fashion as introduced in (Hinton and Salakhutdinov, 2006). The hierarchy stacks one or several feature extraction stages, each of which consists of filter bank layer, non-linear transformation layers, and a pooling layer that combines filter responses over local neighborhoods using an average or max operation, thereby achieving invariance to small distortions. In order to achieve efficient feature extraction and supervised fine-tuning, we only use the predictor module $\mathcal{F}_e(x; K)$ to build the hierarchy.

## 4.1 Modules for Hierarchical Systems

The core component of the feature extraction stage is the filterbank. In conventional systems that contain only a single layer of feature extractor, Gabor functions (Serre et al., 2005; Mutch and Lowe, 2006; Pinto et al., 2008) or oriented edge detectors (Lowe, 2004; Dalal and Triggs, 2005) are the most common choice. In other methods where the feature extraction stage is trained (Ranzato et al., 2007b; Kavukcuoglu et al., 2009; Lee et al., 2009), the first layer filterbank ends up being similar to Gabor-like features, however these methods also provide a method to train successive layers. On the other hand, it is not clear how to design higher level features following Gabor functions.

Another important component of multi-stage architectures is the non-linearities. Convolutional networks (LeCun et al., 1998a) use a simple point-wise sigmoid or *tanh* function following the filterbank, while models that are strongly inspired by biology have included rectifying non-linearities, such as positive part, absolute value, or squaring functions (Pinto et al., 2008), often followed by a local contrast normalization (Pinto et al., 2008), which is inspired by divisive normalization models (Lyu and Simoncelli, 2008).

The final component of a stage is the pooling function that can be applied over space (LeCun et al., 1998a; Lazebnik et al., 2006; Ranzato et al., 2007b; Ahmed et al., 2008), over scale and space (Serre et al., 2005; Mutch and Lowe, 2006; Pinto et al., 2008), or over similar feature types and space (Kavukcuoglu et al., 2009). This layer builds robustness to small distortions by computing an average or a max of the filter responses within the pool.

- **Filter Bank Layer -** $F_{CSG}$: the input of a filter bank layer is a 3D array with $n_1$ 2D *feature maps* of size $n_2 \times n_3$. Each component is denoted $x_{ijk}$, and each feature map is denoted $x_i$. The output is also a 3D array, $y$ composed of $m_1$ feature maps of size $m_2 \times m_3$. A filter in the filter bank $k_{ij}$ has size $l_1 \times l_2$ and connects input feature map $x_i$ to output feature map $y_j$. The module computes the features as defined by equation 4.1.

  $$y_j = \sum_i g_j \cdot \tanh(k_{ij} * x_i) \tag{4.1}$$

  By taking into account the borders effect, we have $m_1 = n_1 - l_1 + 1$, and $m_2 = n_2 - l_2 + 1$. This layer is also denoted by $F_{CSG}$ because it is composed of a set of convolution filters (C), a sigmoid/tanh non-linearity (S), and gain coefficients (G). In the following, superscripts are used to denote the size of the filters. For instance, a filter bank layer with 64 filters of size 9x9, is denoted as: $64F_{CSG}^{9\times9}$. Note that this layer is the predictor function $(\mathcal{F}_e(x; K))$ as defined in chapter 2 applied convolutionally.

- **Rectification Layer -** $R_{abs}$: This module simply applies the absolute value function to all the components of its input: $y_{ijk} = |x_{ijk}|$. Several rectifying non-linearities were tried, including the positive part, and produced similar results.

- **Local Contrast Normalization Layer -** $N$: This module performs local subtractive and divisive normalizations, enforcing a sort of local competition between adjacent features in a feature map, and between features at the same spatial location in different feature maps. The subtractive normalization operation for a given location $x_{ijk}$ computes:

$$v_{ijk} = x_{ijk} - \sum_{ipq} w_{pq} \cdot x_{i,j+p,k+q} \tag{4.2}$$

where $w_{pq}$ is a Gaussian weighting window (of size $9 \times 9$ in our experiments with standard deviation 1.591) normalized so that $\sum_{ipq} w_{pq} = 1$. The divisive normalization computes

$$y_{ijk} = \frac{v_{ijk}}{\max(c, \sigma_{jk})} \tag{4.3}$$

$$\sigma_{jk} = \sqrt{\sum_{ipq} w_{pq} . v^2_{i,j+p,k+q}} \tag{4.4}$$

For each sample, the constant $c$ is set to the $mean(\sigma_{jk})$ in the experiments. The denominator is the weighted standard deviation of all features over a spatial neighborhood. The local contrast normalization layer is inspired by computational neuroscience models (Pinto et al., 2008; Lyu and Simoncelli, 2008).

- **Average Pooling and Subsampling Layer -** $P_A$: The purpose of this layer is to build robustness to small distortions, playing the same role as the complex cells in models of visual perception. Each output value is

$$y_{ijk} = \sum_{pq} w_{pq} \cdot x_{i,j+p,k+q} \tag{4.5}$$

where $w_{pq}$ is a uniform weighting window ("boxcar filter"). Each output feature map is then subsampled spatially by a factor $S$ horizontally and vertically.

- **Max-Pooling and Subsampling Layer -** $P_M$: building local invariance to shift can be performed with any symmetric pooling operation. The max-pooling module is similar to the average pooling, except that the average operation is replaced by a max operation. In our experiments, the pooling windows were non-overlapping. A max-pooling layer with 4x4 down-sampling is denoted $P_M^{4\times 4}$.

## 4.2 Combining Modules into a Hierarchy

Different architectures can be produced by cascading the above-mentioned modules in various ways. An architecture is composed of one or two stages of feature extraction, each of which is formed by cascading a filtering layer with different combinations of rectification, normalization, and pooling. Recognition architectures are composed of one or two such stages, followed by a classifier, generally a multinomial logistic regression. A single stage with filterbank, rectification, contrast normalization and pooling stages are shown in Fig. 4.1.

- $\mathbf{F_{CSG} - P_A}$ This is the basic building block of traditional convolutional networks, alternating tanh-squashed filter banks with average down-sampling layers (LeCun et al., 1998a; Huang and LeCun, 2006). A complete convolu-

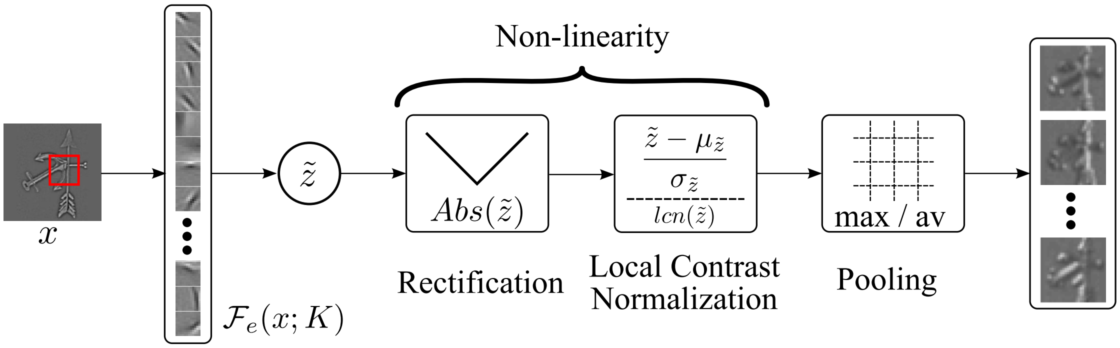

Figure 4.1: An example of a feature extraction stage of the type $F_{CSG} - R_{abs} - N - P_A$. An input image (or a feature map) is passed through a non-linear filterbank, followed by rectification, local contrast normalization and spatial pooling/sub-sampling.

tional network would have several sequences of "$F_{CSG}$ - $P_A$" followed by a linear classifier.

- $\mathbf{F_{CSG} - R_{abs} - P_A}$ The tanh-squashed filter bank is followed by an absolute value non-linearity, and by an average down-sampling layer.

- $\mathbf{F_{CSG} - R_{abs} - N - P_A}$ The tanh-squashed filter bank is followed by an absolute value non-linearity, by a local contrast normalization layer and by an average down-sampling layer.

- $\mathbf{F_{CSG} - P_M}$ This is also a typical building block of convolutional networks, as well as the basis of the HMAX and other architectures (Serre et al., 2005; Ranzato et al., 2007b), which alternate tanh-squashed filter banks with max-pooling layers.

## 4.3 Training Protocol

Given a particular architecture, a number of training protocols have been considered and tested. Each protocol is identified by a letter $R, U, R^+$, or $U^+$. A single letter (e.g. $R$) indicates an architecture with a single stage of feature extraction, followed by a classifier, while a double letter (e.g. $RR$) indicates an architecture with two stages of feature extraction followed by a classifier. $R$ and $RR$ represents feature extraction using random kernels. $U$ and $UU$ represents feature extraction using kernels initialized using the PSD algorithm. $R$ and $U$ followed by a superscript $^+$ represents supervised training of the combined feature extractor and classifier.

- **Random Features and Supervised Classifier - R and RR**: The filters in the feature extraction stages are *set to random values and kept fixed* (no feature learning takes place), and the classifier stage is trained in supervised mode.

- **Unsupervised Features, Supervised Classifier - U and UU**. The predictor ($\mathcal{F}_e$) filters of the feature extraction stages are trained using the *unsupervised* PSD algorithm, described in chapter 2, and kept fixed. The classifier stage is trained in supervised mode.

- **Random Features, Global Supervised Refinement - $\mathbf{R^+}$ and $\mathbf{R^+R^+}$.** The filters in the feature extractor stages are *initialized with random values*, and the *entire system (feature stages + classifier) is trained in supervised mode by gradient descent*. The gradients are computed using back-propagation, and all the filters are adjusted by stochastic online updates. This is identical to the usual method for training supervised convolutional networks.

- **Unsupervised Feature, Global Supervised Refinement - $\mathbf{U^+}$ and $\mathbf{U^+U^+}$.** The filters in the feature extractor stages are *initialized with the PSD unsupervised learning algorithm*, and the *entire system (feature stages + classifier) is then trained (refined) in supervised mode by gradient descent*. The system is trained the same way as *random features with global refinement* using online stochastic updates. This is reminiscent of the "deep belief network" strategy in which the stages are first trained in unsupervised mode one after the other, and then globally refined using supervised learning (Hinton and Salakhutdinov, 2006; Bengio et al., 2007; Ranzato et al., 2007a).

A traditional convolutional network with a single stage initialized at random (LeCun et al., 1998a) would be denoted by an architecture motif like "$F_{CSG} - P_A$", and the training protocol would be denoted by $R^+$. The stages of a convolutional network with max-pooling would be denoted by "$F_{CSG} - P_M$". A system with two such stages trained in unsupervised mode, and the classifier (only) trained in supervised mode, as in (Ranzato et al., 2007b), is denoted $UU$ and a two stage "deep belief network" is represented as $\mathbf{U}^+\mathbf{U}^+$.

## 4.4 Experiments with Caltech-101 Dataset

We have used Caltech-101 (Fei-Fei et al., 2004) dataset to compare different training protocols and architectures. A more detailed study can be found in our work *"What is the best multi-stage architecture for object recognition?"* (Jarrett et al., 2009).

Images from the Caltech-101 dataset were pre-processed with a procedure similar to (Pinto et al., 2008). The steps are: 1) converting to gray-scale (no color) and resizing to $151 \times 151$ pixels. 2) subtracting the image mean and dividing by the image standard deviation, 3) applying subtractive/divisive normalization ($N$ layer with $c = 1$). 4) zero-padding the shorter side to 143 pixels.

### 4.4.1 Using a Single Stage of Feature Extraction

The first stage is composed of an $F_{CSG}$ layer with 64 filters of size $9{\times}9$ ($64F_{CSG}^{9\times 9}$), followed by an abs rectification ($R_{abs}$), a local contrast normalization layer ($N$) and an average pooling layer with $10 \times 10$ boxcar filter and $5 \times 5$ down-sampling ($P_A^{5\times 5}$). The output of the first stage is a set of 64 features maps of size $26 \times 26$. This output is then fed to a multinomial logistic regression classifier that produces a 102-dimensional output vector representing a posterior distribution over class labels. Lazebnik's PMK-SVM classifier (Lazebnik et al., 2006) was also tested.

<table>
<tr><th colspan="6">Linear Logistic Regression Classifier</th></tr>
<tr><th colspan="6">Single Stage $[\mathbf{64.F^{9\times9}_{CSG} - R/N/P^{5\times5}}]$</th></tr>
<tr><td></td><td>$\mathbf{R_{abs}/N/P_A}$</td><td>$\mathbf{R_{abs}/P_A}$</td><td>$\mathbf{N/P_M}$</td><td>$\mathbf{N/P_A}$</td><td>$\mathbf{P_A}$</td></tr>
<tr><td>$\mathbf{U^+}$</td><td>54.2%</td><td>50.0%</td><td>44.3%</td><td>18.5%</td><td>14.5%</td></tr>
<tr><td>$\mathbf{R^+}$</td><td>54.8%</td><td>47.0%</td><td>38.0%</td><td>16.3%</td><td>14.3%</td></tr>
<tr><td><b>U</b></td><td>52.2%</td><td>43.3%(±1.6)</td><td>44.0%</td><td>17.2%</td><td>13.4%</td></tr>
<tr><td><b>R</b></td><td>53.3%</td><td>31.7%</td><td>32.1%</td><td>15.3%</td><td>12.1%(±2.2)</td></tr>
<tr><th colspan="6">Two Stages $[\mathbf{64.F^{9\times9}_{CSG} - R/N/P^{5\times5}}] - [\mathbf{256.F^{9\times9}_{CSG} - R/N/P^{4\times4}}]$</th></tr>
<tr><td></td><td>$\mathbf{R_{abs}/N/P_A}$</td><td>$\mathbf{R_{abs}/P_A}$</td><td>$\mathbf{N/P_M}$</td><td>$\mathbf{N/P_A}$</td><td>$\mathbf{P_A}$</td></tr>
<tr><td>$\mathbf{U^+U^+}$</td><td>65.5%</td><td>60.5%</td><td>61.0%</td><td>34.0%</td><td>32.0%</td></tr>
<tr><td>$\mathbf{R^+R^+}$</td><td>64.7%</td><td>59.5%</td><td>60.0%</td><td>31.0%</td><td>29.7%</td></tr>
<tr><td><b>UU</b></td><td>63.7%</td><td>46.7%</td><td>56.0%</td><td>23.1%</td><td>9.1%</td></tr>
<tr><td><b>RR</b></td><td>62.9%</td><td>33.7%(±1.5)</td><td>37.6%(±1.9)</td><td>19.6%</td><td>8.8%</td></tr>
<tr><th colspan="6">PMK-SVM Classifier</th></tr>
<tr><th colspan="6">Single Stage: $[\mathbf{64.F^{9\times9}_{CSG} - R_{abs}/N/P^{5\times5}_{A}}]$</th></tr>
<tr><td><b>U</b></td><td>64.0%</td><td colspan="4"></td></tr>
<tr><th colspan="6">Two Stages: $[\mathbf{64.F^{9\times9}_{CSG} - R_{abs}/N/P^{5\times5}_{A}}] - [\mathbf{256.F^{9\times9}_{CSG} - R_{abs}/N}]$</th></tr>
<tr><td><b>UU</b></td><td>52.8%</td><td colspan="4"></td></tr>
</table>

Table 4.1: Average recognition rates on Caltech-101 with 30 training samples per class. Each row contains results for one of the training protocols, and each column for one type of architecture. All columns use an $F_{CSG}$ as the first module, followed by the modules shown in the column label. The error bars for all experiments are within 1%, except where noted.

### 4.4.2 Using Two Stages of Feature Extraction

In two-stage systems, the second-stage feature extractor is fed with the output of the first stage. The first layer of the second stage is an $F_{CSG}$ module with 256 output features maps, each of which combines a random subset of 16 feature maps from the previous stage using $9 \times 9$ kernels. Hence the total number of convolution kernels is $256 \times 16 = 4096$. The average pooling module uses a $6 \times 6$ boxcar filter with a $4 \times 4$ down-sampling step. This produces an output feature map of size $256 \times 4 \times 4$, which is then fed to a multinomial logistic regression classifier. The PMK-SVM classifier was also tested.

Results for these experiments are summarized in Table 4.1.

1. The most astonishing result is that systems with *random filters and no filter learning whatsoever achieve decent performance* (53.3% for $R$ and 62.9% for RR), as long as they include absolute value rectification and contrast normalization ($R_{abs} - N - P_A$).

2. Comparing experiments from rows $R$ vs $R^+$, $RR$ vs $R^+R^+$, $U$ vs $U^+$ and $UU$ vs $U^+U^+$, we see that supervised fine tuning consistently improves the performance, particularly with weak non-linearities: 62.9% to 64.7% for $RR$, 63.7% to 65.5% for $UU$ using $R_{abs} - N - P_A$ and 35.1% to 59.5% for $RR$ using $R_{abs} - P_A$.

3. It appears clear that two-stage systems ($UU$, $U^+U^+$, $RR$, $R^+R^+$) are systematically and significantly better than their single-stage counterparts ($U$, $U^+$,

$R$, $R^+$). For instance, 54.2% obtained by $U^+$ compared to 65.5% obtained by $U^+U^+$ using $R_{abs} - N - P_A$. However, when using the $P_A$ architecture, adding a second stage without supervised refinement does not seem to help. This may be due to cancellation effects of the $P_A$ layer when rectification is not present.

4. It seems that unsupervised training ($U$, $UU$, $U^+$, $U^+U^+$) does not seem to significantly improve the performance (comparing with ($R$, $RR$, $R^+$, $R^+R^+$) if both rectification and normalization are used (62.9% for $RR$ versus 63.7% for $UU$). When contrast normalization is removed, the performance gap becomes significant (33.7% for $RR$ versus 46.7% for $UU$). If no supervised refinement is performed, it looks as if appropriate architectural components are a good substitute for unsupervised training.

5. It is clear that abs rectification is a crucial component for achieving good performance, as shown with the $U^+U^+$ protocol by comparing columns $P_A$ (32.0%) and $R_{abs} - N - P_A$ (65.5%). However, using max pooling seems to alleviate the need for abs rectification, confirming the hypothesis that average pooling without rectification falls victim to cancellation effects between neighboring filter outputs.

6. A single-stage system with PMK-SVM reaches the same performance as a two-stage system with logistic regression (around 65%) as shown in the last two rows of Table 4.1. It looks as if the pyramid match kernel is able to play

the same role as a second stage of feature extraction. Perhaps it is because PMK first performs a K-means based vector quantization, which can be seen as an extreme form of sparse coding, followed by local histogramming, which is a form of spatial pooling. Hence, the PM kernel is conceptually similar to a second stage based on sparse coding and pooling as recently pointed out in (Yang et al., 2009). Furthermore, these numbers are similar to the performance of the original PMK-SVM system which used dense SIFT features (64.6%) (Lazebnik et al., 2006). Again, this is hardly surprising as the SIFT module is conceptually very similar to our feature extraction stage. When using features extracted using $UU$ architecture, the performance of PMK-SVM classifier drops significantly. This behaviour might be caused by the very small spatial density ($18 \times 18$) of features at the second layer.

## 4.5 NORB Dataset

Caltech-101 is very peculiar in that many objects have distinctive textures and few pose variations. More importantly, the particularly small size of the training set favors methods that minimize the role learning and maximize the role of clever engineering. A diametrically opposed object dataset is NORB (LeCun et al., 2004). The “Small NORB” dataset has 5 object categories (humans, airplanes, cars, trucks, animals) 5 object instances for training, and 5 different object instances for test. Each object instance has 972 samples (18 azimuths, 9 elevations, and 6 illuminations), for a total of 24300 training samples and 24300 test samples

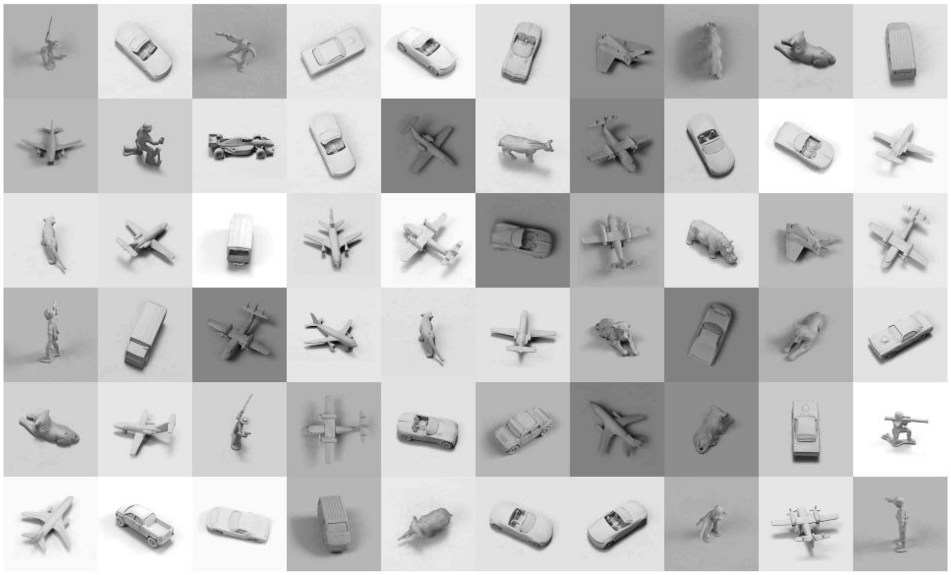

Figure 4.2: Several examples from NORB dataset.

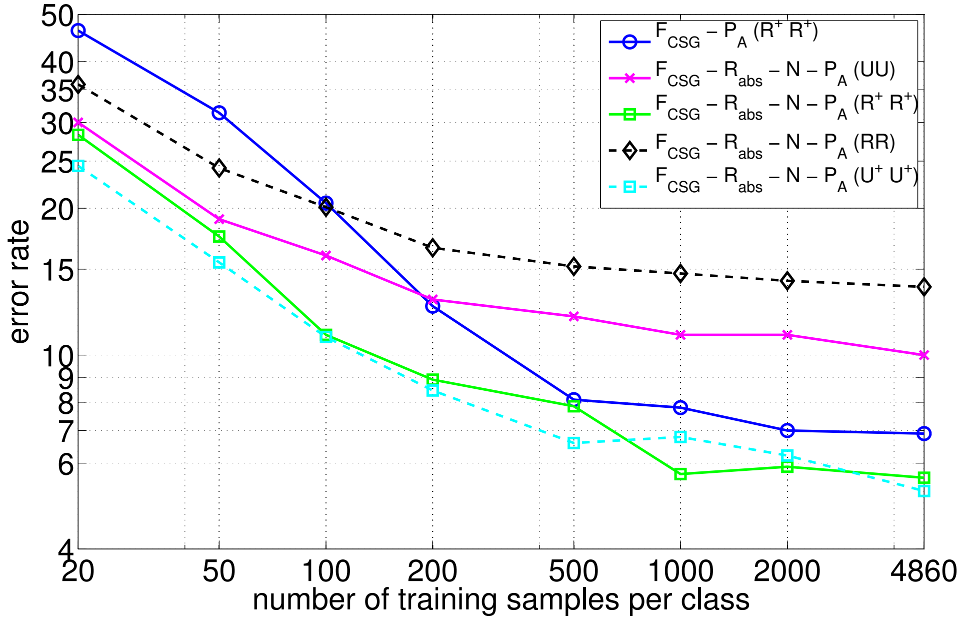

Figure 4.3: Test Error rate vs. number of training samples per class on NORB Dataset. Although pure random features perform surprisingly well when training data is very scarce, for large number of training data learning improves the performance significantly. Absolute value rectification ($R_{abs}$) and local normalization ($N$) is shown to improve the performance in all cases.

(4860 per class). Each image is $96 \times 96$ pixels, grayscale. Experiments were conducted to elucidate the importance of the non-linearity, and the performance of random filter systems when many labeled samples are available.

Only the $RR$, $R^+R^+$, $UU$, $U^+U^+$ protocols were used with 8 feature maps and $5 \times 5$ filters at the first stage, $4 \times 4$ average pooling followed by 24 feature maps with $6 \times 6$ filters, each of which combines input from 4 randomly picked stage-1 feature maps, followed by $3 \times 3$ average pooling. The last stage is a 5-way multinomial logistic regressor.

Several systems with various non-linearities were trained on subsets of the training set with 20, 50, 100, 200, 500, 1000, 2000, and 4860 training samples per class. The results are shown in figure 4.3 in log-log scale. All the curves except the blue curve use represent models using absolute value rectification and contrast normalization. It appears that the use of abs and normalization makes a big difference when labeled samples are scarce, but the difference diminishes as the number of training samples increases. Training seems to compensate for architectural simplicity, or conversely architectural sophistication seems to compensate for lack of training. Comparing $UU$ and $RR$, it is apparent that unsupervised learning improves the recognition performance significantly, however, $R^+R^+$ shows that labeled information is more effective than unsupervised information. Moreover, comparing $R^+R^+$ and $U^+U^+$, it can be seen that unsupervised training is most effective when number of samples is not large ($< 100$). The best error rate when trained on the full training set with a model including abs and normalization is

5.6% and 5.5% for $R^+R^+$ and $U^+U^+$ respectively, but 6.9% with neither ((LeCun et al., 2004) reported 6.6%).

More interesting is the behavior of the system with random filters: While its error rate is comparable to that of the network trained in supervised mode for small training sets (in the "Caltech-101 regime"), the error rate remains high as samples are added. Hence, while random filters perform well on Caltech-101, they would most likely not perform as well as learned filters on tasks with more labeled samples.

## 4.6 Random Filter Performance

The NORB experiments show that random filters yield sub-par performance when labeled samples are abundant. But the experiments also show that random filters seem to require the presence of abs and normalization. To explore why random filters work at all, starting from randomly initialized inputs, we used gradient descent minimization to find the *optimal input patterns* that maximize each output unit (after pooling) in a $F_{CSG} - R_{abs} - N - P_A$ stage. The surprising finding is that the optimal stimuli for random filters are oriented gratings (albeit noisy and faint ones), similar to the optimal stimuli for trained filters. As shown in fig 4.4, it appears that random weights, combined with the abs/norm/pooling creates a spontaneous orientation selectivity.

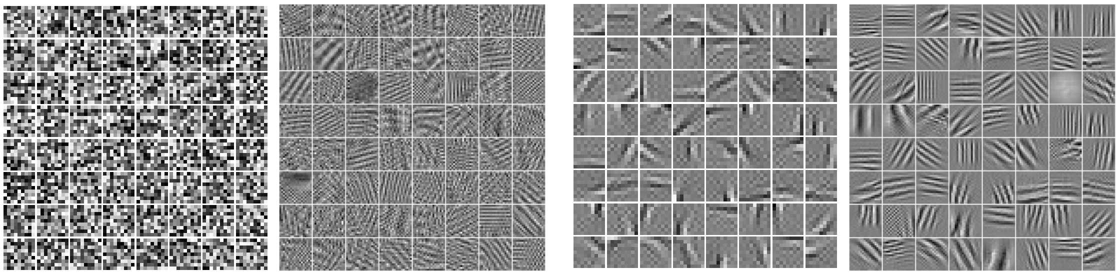

Figure 4.4: Left: random stage-1 filters, and corresponding optimal inputs that maximize the response of each corresponding complex cell in a $F_{CSG} - R_{abs} - N - P_A$ architecture. The small asymmetry in the random filters is sufficient to make them orientation selective. Right: same for PSD filters. The optimal input patterns contain several periods since they maximize the output of a complete stage that contains rectification, local normalization, and average pooling with down-sampling. Shifted versions of each pattern yield similar activations.

## 4.7 Handwritten Digits Recognition

As a sanity check for the overall training procedures and architectures, experiments were run on the MNIST dataset, which contains 60,000 gray-scale 28x28 pixel digit images for training and 10,000 images for testing. An architecture with two stages of feature extraction was used: the first stage produces 32 feature maps using $5 \times 5$ filters, followed by 2x2 average pooling and down-sampling. The second stage produces 64 feature maps, each of which combines 16 feature maps from stage 1 with $5 \times 5$ filters (1024 filters total), followed by $2 \times 2$ pooling/down-sampling. The classifier is a 2-layer fully-connected neural network with 200 hidden units, and 10 outputs. The loss function is equivalent to that of a 10-way multinomial logistic regression (also known as cross-entropy loss). The two feature stages use abs rectification and normalization.

The parameters for the two feature extraction stages are first trained with PSD as explained in Section 2. The classifier is initialized randomly. The whole system is fine-tuned in supervised mode (the protocol could be described as $(U^+U^+R^+R^+)$. A validation set of size 10,000 was set apart from the training set to tune the only hyper-parameter: the sparsity constant $\lambda$. Nine different values were tested between 0.1 and 1.6 and the best value was found to be 0.2. The system was trained with stochastic gradient descent on the 50,000 non-validation training samples until the best error rate on the validation set was reached (this took 30 epochs). We also used a diagonal approximation to the Hessian of the parameters using Levenberg-Marquardt method. We updated the Hesssian approximation at every epoch using 500 samples. The whole system was then tuned for another 3 epochs on the whole training set. A **test error rate of 0.53% was obtained**. To our knowledge, *this is the best error rate ever reported on the original MNIST dataset, without distortions or preprocessing*. The best previously reported error rate was 0.60% (Ranzato et al., 2006).

# Chapter 5

# Convolutional Sparse Coding

Although sparse modeling has been used successfully for image analysis (Aharon et al., 2005; Mairal et al., 2009) and learning feature extractors (Kavukcuoglu et al., 2009), these systems are trained on *single image patches* whose dimensions match those of the filters. In sparse coding, a sparse feature vector $z$ is computed so as to best reconstruct the input $x$ through a linear operation with a *fixed dictionary matrix* $\mathcal{D}$. The inference procedure produces a code $z^*$ by minimizing an energy function:

$$E(x, z, \mathcal{D}) = \frac{1}{2}||x - \mathcal{D}z||_2^2 + \lambda|z|_1, \quad z^* = \arg\min_z E(x, z, \mathcal{D}) \tag{5.1}$$

As explained in Chapter 2, the dictionary can be learned by minimizing the energy 5.1 wrt $\mathcal{D}$: $\min_{z,\mathcal{D}} E(x, z, \mathcal{D})$ averaged over a training set of input samples. After training, patches in the image are processed separately. This procedure completely ignores the fact that the filters are eventually going to be used in a convolutional fashion. Learning will produce a dictionary of filters that are essentially shifted versions of each other over the patch, so as to reconstruct each patch in isolation. Inference is performed on all (overlapping) patches independently, which produces a very highly redundant representation for the whole image.

To address this problem, we apply sparse coding to the entire image at once, and we view the dictionary as a convolutional filter bank:

$$E(x, z, \mathcal{D}) = \frac{1}{2}||x - \sum_i \mathcal{D}_i * z_i||_2^2 + \lambda|z|_1, \tag{5.2}$$

where $\mathcal{D}_i$ is an $s \times s$ 2D filter kernel, $x$ is a $w \times h$ image (instead of an $s \times s$ patch), $z_i$ is a 2D feature map of dimension $(w - s + 1) \times (h - s + 1)$, and "$*$" denotes the discrete convolution operator. Convolutional Sparse Coding has been used by several authors, including (Zeiler et al., 2010).

Utilizing same idea proposed in PSD , we can modify the convolutional sparse coding formulation in eq. (5.2) to include a trainable feed-forward predictor $\mathcal{F}_e$.

$$E(x, z, \mathcal{D}, K) = \frac{1}{2}||x - \sum_i \mathcal{D}_i * z_i||_2^2 + \lambda|z|_1 + \alpha \sum_i ||z_i - \mathcal{F}_e(x; k^i, g^i)||_2^2 \tag{5.3}$$

where $z^* = \arg\min_z \mathcal{L}(x, z, \mathcal{D}, K)$, $K = \{k, g\}$ and $k^i$ is an encoding convolution kernel of size $s \times s$, $g^i$ is a singleton per feature map $i$ and $\mathcal{F}_e$ is the encoder function. Two crucially important questions are the form of the $\mathcal{F}_e$, and the optimization method to find $z^*$. Both questions will be discussed at length below.

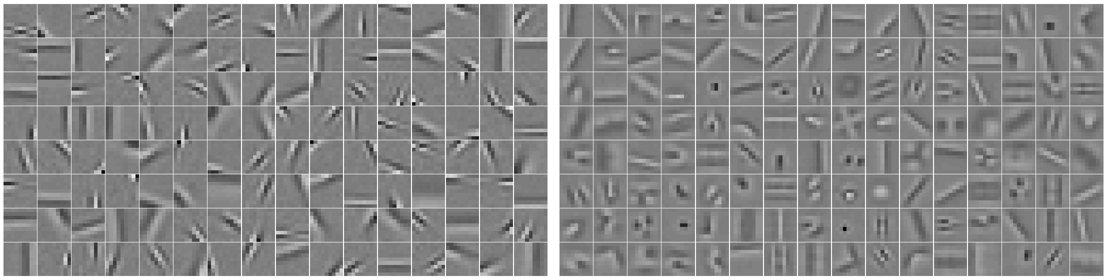

Figure 5.1: **Left:** A dictionary with 128 elements, learned with patch based sparse coding model. **Right:** A dictionary with 128 elements, learned with convolutional sparse coding model. The dictionary learned with the convolutional model spans the orientation space much more uniformly. In addition it can be seen that the diversity of filters obtained by convolutional sparse model is much richer compared to patch based one.

## 5.1 Algorithms and Method

In this section, we analyze the benefits of convolutional sparse coding for object recognition systems, and propose convolutional extensions to the coordinate descent sparse coding (CoD) (Li and Osher) algorithm and the dictionary learning procedure.

### 5.1.1 Learning Convolutional Dictionaries

The key observation for modeling convolutional filter banks is that the convolution of a signal with a given kernel can be represented as a matrix-vector product by constructing a special Toeplitz-structured matrix for each dictionary element and concatenating all such matrices to form a new dictionary. Any existing sparse coding algorithm can then be used. Unfortunately, this method incurs a cost, since the size of the dictionary then depends on the size of the input signal. Therefore, it is advantageous to use a formulation based on convolutions rather than following the naive method outlined above. In this work, we use the coordinate descent sparse coding algorithm (Li and Osher) as a starting point and generalize it using convolution operations. Two important issues arise when learning convolutional dictionaries: 1. The boundary effects due to convolutions need to be properly handled. 2. The derivative of equation 5.2 should be computed efficiently. Since the loss is not jointly convex in $\mathcal{D}$ and $z$, but is convex in each variable when the other one is kept fixed, sparse dictionaries are usually learned by an approach similar to block coordinate descent, which alternatively minimizes over $z$ and $\mathcal{D}$ (e.g., see (Olshausen and Field, 1997; Mairal et al., 2009; Ranzato et al., 2006, 2007a)). One can use either batch (Aharon et al., 2005) (by accumulating derivatives over many samples) or online updates (Mairal et al., 2009; Zeiler et al., 2010; Kavukcuoglu et al., 2009) (updating the dictionary after each sample). In this work, we use a stochastic online procedure for updating the dictionary elements.

The updates to the dictionary elements, calculated from equation 5.2, are sensitive to the boundary effects introduced by the convolution operator. The code units that are at the boundary might grow much larger compared to the middle elements, since the outermost boundaries of the reconstruction take contributions from only a single code unit, compared to the middle ones that combine $s \times s$ units.

Therefore the reconstruction error, and correspondingly the derivatives, grow proportionally larger. One way to properly handle this situation is to apply a mask on the derivatives of the reconstruction error wrt $z$: $D^T * (x - \mathcal{D} * z)$ is replaced by $D^T * (mask(x) - \mathcal{D} * z)$, where $mask$ is a term-by-term multiplier that either puts zeros or gradually scales down the boundaries.

**Algorithm 3** Convolutional extension to coordinate descent sparse coding(Li and Osher). A subscript index (set) of a matrix represent a particular element. For slicing the $4D$ tensor $S$ we adopt the MATLAB notation for simplicity of notation.

```
function ConvCoD($x, \mathcal{D}, \alpha$)
    Set: $S = \mathcal{D}^T * \mathcal{D}$
    Initialize: $z = 0$; $\beta = \mathcal{D}^T * mask(x)$
    Require: $h_\alpha$ : smooth thresholding function.
    repeat
        $\bar{z} = h_\alpha(\beta)$
        $(k, p, q) = \arg\max_{i,m,n} |z_{imn} - \bar{z}_{imn}|$ ($k$ : dictionary index, $(p.q)$ : location index)
        $bi = \beta_{kpq}$
        $\beta = \beta + (z_{kpq} - \bar{z}_{kpq}) \times align(S(:, k, :, :), (p, q))$
        $z_{kpq} = \bar{z}_{kpq}$, $\beta_{kpq} = bi$
    until change in $z$ is below a threshold
end function
```

The second important point in training convolutional dictionaries is the computation of the $S = D^T * D$ operator. For most algorithms like coordinate descent (Li and Osher), FISTA (Beck and Teboulle, 2009) and matching pursuit (Mallat and Zhang, 1993), it is advantageous to store the similarity matrix ($S$) explicitly and use a single column at a time for updating the corresponding component of code $z$. For convolutional modeling, the same approach can be followed with some additional care. In patch based sparse coding, each element $(i, j)$ of $S$ equals the dot product of dictionary elements $i$ and $j$. Since the similarity of a pair of dictionary elements has to be also considered in spatial dimensions, each term is expanded as *"full"* convolution of two dictionary elements $(i, j)$, producing $2s - 1 \times 2s - 1$ matrix. It is more convenient to think about the resulting matrix as a $4D$ tensor of size $d \times d \times 2s - 1 \times 2s - 1$. One should note that, depending on the input image size, proper alignment of corresponding column of this tensor has to be applied in the $z$ space. One can also use the steepest descent algorithm for finding the solution to convolutional sparse coding given in equation 5.2, however using this method would be orders of magnitude slower compared to specialized algorithms like CoD (Li and Osher) and the solution would never contain exact zeros. In algorithm 3 we explain the extension of the coordinate descent algorithm (Li and Osher) for convolutional inputs. Having formulated convolutional sparse coding, the overall learning procedure is simple stochastic (online) gradient descent over dictionary $\mathcal{D}$:

$$\forall x^i \in \mathcal{X} \ \text{ training set : } \quad z^* = \arg\min_z \mathcal{L}(x^i, z, \mathcal{D}), \ \ \mathcal{D} \leftarrow \mathcal{D} - \eta \frac{\partial \mathcal{L}(x^i, z^*, \mathcal{D})}{\partial \mathcal{D}} \tag{5.4}$$

The columns of $\mathcal{D}$ are normalized after each iteration. A convolutional dictionary with 128 elements which was trained on images from Berkeley dataset (Martin et al., 2001) is shown in figure 5.1.

### 5.1.2 Learning an Efficient Encoder

In (Ranzato et al., 2007a), (Jarrett et al., 2009) and (Gregor and LeCun, 2010) a feedforward regressor was trained for fast approximate inference. In this work, we extend their encoder module training to convolutional domain and also propose a new encoder function that approximates sparse codes more closely. The encoder used in (Jarrett et al., 2009) is a simple feedforward function which can also be seen as a small convolutional neural network: $\tilde{z} = g^i \times tanh(x * k^i)\ \ (i = 1..d)$. This function has been shown to produce good features for object recognition (Jarrett et al., 2009), however it does not include a shrinkage operator, thus its ability to produce sparse representations is very limited. Therefore, we propose a different encoding function with a shrinkage operator. The standard soft thresholding operator has the nice property of producing exact zeros around the origin, however for a very wide region, the derivatives are also zero. In order to be able to train a filter bank that is applied to the input before the shrinkage operator, we propose to use an encoder with a smooth shrinkage operator $\tilde{z} = sh_{\beta^i,b^i}(x * k^i)$ where $i = 1..d$ and :

$$sh_{\beta^i,b^i}(s) = sign(s) \times 1/\beta^i \log(\exp(\beta^i \times b^i) + \exp(\beta^i \times |s|) - 1) - b^i \tag{5.5}$$

Note that each $\beta^i$ and $b^i$ is a singleton per each feature map $i$. The shape of the smooth shrinkage operator is given in figure 5.2 for several different values of $\beta$ and

$b$. It can be seen that $\beta$ controls the smoothness of the kink of shrinkage operator and $b$ controls the location of the kink. The function is guaranteed to pass through the origin and is antisymmetric. The partial derivatives $\frac{\partial sh}{\partial \beta}$ and $\frac{\partial sh}{\partial b}$ can be easily written and these parameters can be learned from data.

Updating the parameters of the encoding function is performed by minimizing equation 5.3. The additional cost term penalizes the squared distance between optimal code $z$ and prediction $\tilde{z}$. In a sense, training the encoder module is similar to training a ConvNet. To aid faster convergence, we use stochastic diagonal Levenberg-Marquardt method (LeCun et al., 1998b) to calculate a positive diagonal approximation to the hessian. We update the hessian approximation every 10000 samples and the effect of hessian updates on the total loss is shown in figure 5.2. It can be seen that especially for the *tanh* encoder function, the effect of using second order information on the convergence is significant.

### 5.1.3 Patch Based vs Convolutional Sparse Modeling

Natural images, sounds, and more generally, signals that display translation invariance in any dimension, are better represented using convolutional dictionaries. The convolution operator enables the system to model local structures that appear anywhere in the signal. For example, if $m \times m$ image patches are sampled from a set of natural images, an edge at a given orientation may appear at any location, forcing local models to allocate multiple dictionary elements to represent a single underlying orientation. By contrast, a convolutional model only needs to record

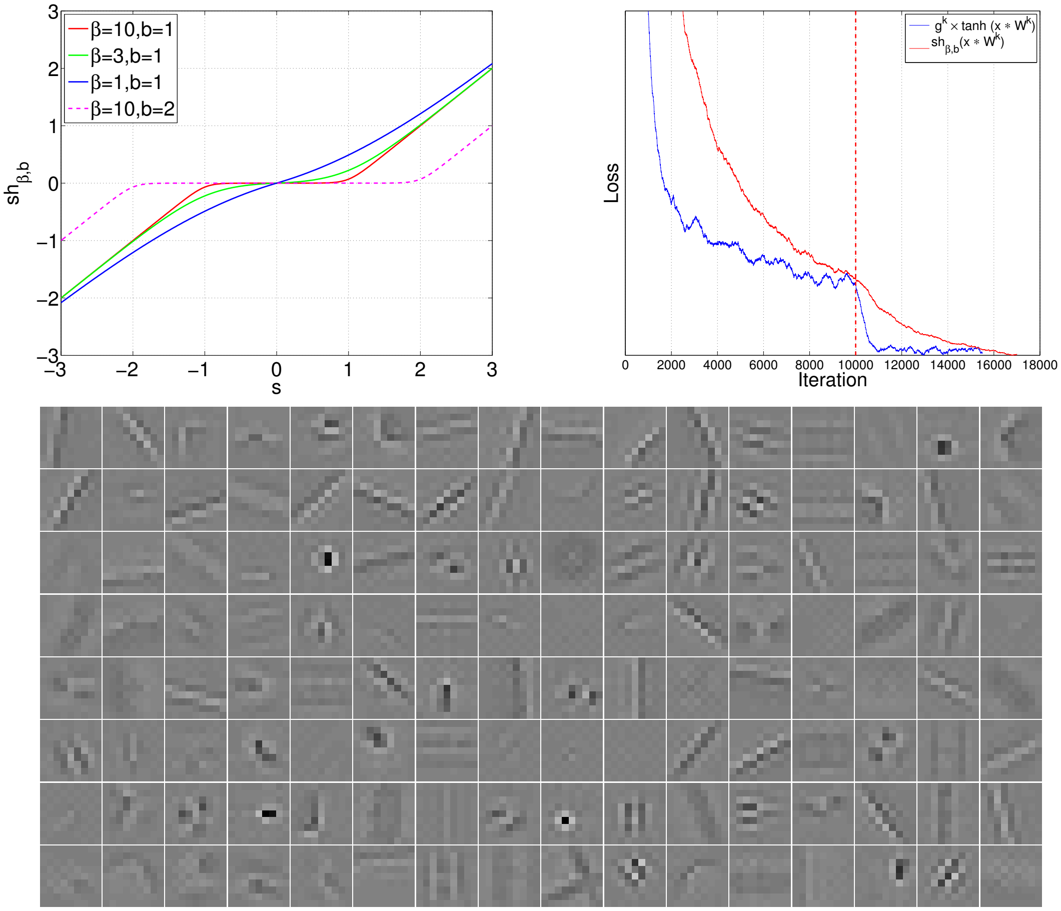

Figure 5.2: **Top Left:** Smooth shrinkage function. Parameters $\beta$ and $b$ control the smoothness and location of the kink of the function. As $\beta \to \infty$ it converges more closely to soft thresholding operator. **Top Right:** Total loss as a function of number of iterations. The vertical dotted line marks the iteration number when diagonal hessian approximation was updated. It is clear that for both encoder functions, hessian update improves the convergence significantly. **Bottom:** 128 convolutional filters $(k)$ learned in the encoder using smooth shrinkage function. The decoder of this system is shown in image 5.1.

the oriented structure once, since dictionary elements can be used at all locations. Figure 5.1 shows atoms from patch-based and convolutional dictionaries comprising the same number of elements. The convolutional dictionary does not waste resources modeling similar filter structure at multiple locations. Instead, it models more orientations, frequencies, and different structures including center-surround filters, double center-surround filters, and corner structures at various angles.

In this work, we present two encoder architectures, 1. steepest descent sparse coding with *tanh* encoding function using $g^i \times tanh(x * k^i)$, 2. convolutional CoD sparse coding with *shrink* encoding function using $sh_{\beta,b}(x * k^i)$. The time required for training the first system is much higher than for the second system due to steepest descent sparse coding. However, the performance of the encoding functions are almost identical.

### 5.1.4 Multi-stage architecture

Our convolutional encoder can be used to replace patch-based sparse coding modules used in multi-stage object recognition architectures such as the one proposed in our previous work (Jarrett et al., 2009). Building on our previous findings, for each stage, the encoder is followed by and absolute value rectification, contrast normalization and average subsampling. **Absolute Value Rectification** is a simple pointwise absolute value function applied on the output of the encoder. **Contrast Normalization** is the same operation used for pre-processing the images. This type of operation has been shown to reduce the dependencies between components (Schwartz and Simoncelli, 2001; Lyu and Simoncelli, 2008) (feature maps in our case). When used in between layers, the mean and standard deviation is calculated across all feature maps with a $9 \times 9$ neighborhood in spatial dimensions. The last operation, **average pooling** is simply a spatial pooling operation that is applied on each feature map independently.

One or more additional stages can be stacked on top of the first one. Each stage then takes the output of its preceding stage as input and processes it using the same series of operations with different architectural parameters like size and connections. When the input to a stage is a series of feature maps, each output feature map is formed by the summation of multiple filters.

In the next sections, we present experiments showing that using convolutionally trained encoders in this architecture lead to better object recognition performance.

## 5.2 Experiments

We closely follow the architecture proposed in (Jarrett et al., 2009) for object recognition experiments. As stated above, in our experiments, we use two different systems: **1.** Steepest descent sparse coding with *tanh* encoder: $\mathbf{SD}^{tanh}$. **2.** Coordinate descent sparse coding with *shrink* encoder: $\mathbf{CD}^{shrink}$. In the following, we give details of the unsupervised training and supervised recognition experiments.

### 5.2.1 Object Recognition using Caltech 101 Dataset

The Caltech-101 dataset (Fei-Fei et al., 2004) contains up to 30 training images per class and each image contains a single object. We process the images in the dataset as follows: **1.** Each image is converted to gray-scale and resized so that the largest edge is 151. **2.** Images are contrast normalized to obtain locally zero mean and unit standard deviation input using a $9 \times 9$ neighborhood. **3.** The short side of each image is zero padded to 143 pixels. We report the results in Table 5.1 and 5.2. All results in these tables are obtained using 30 training samples per class and 5 different choices of the training set. We use the background class during training and testing.

**Architecture :** We use the unsupervised trained encoders in a multi-stage system identical to the one proposed in (Jarrett et al., 2009). At first layer 64 features are extracted from the input image, followed by a second layers that produces 256 features. Second layer features are connected to fist layer features through a sparse connection table to break the symmetry and to decrease the number of parameters.

**Unsupervised Training :** The input to unsupervised training consists of contrast normalized gray-scale images (Pinto et al., 2008) obtained from the Berkeley segmentation dataset (Martin et al., 2001). Contrast normalization consists of processing each feature map value by removing the mean and dividing by the standard deviation calculated around $9 \times 9$ region centered at that value over all feature maps.

**First Layer:** We have trained both systems using 64 dictionary elements. Each dictionary item is a $9 \times 9$ convolution kernel. The resulting system to be solved is a 64 times overcomplete sparse coding problem. Both systems are trained for 10 different sparsity values ranging between 0.1 and 3.0.

**Second Layer:** Using the 64 feature maps output from the first layer encoder on Berkeley images, we train a second layer convolutional sparse coding. At the second layer, the number of feature maps is 256 and each feature map is connected to 16 randomly selected input features out of 64. Thus, we aim to learn 4096 convolutional kernels at the second layer. To the best of our knowledge, none of the previous convolutional RBM (Lee et al., 2009) and sparse coding (Zeiler et al., 2010) methods have learned such a large number of dictionary elements. Our aim is motivated by the fact that using such large number of elements and using a linear classifier  (Jarrett et al., 2009) reports recognition results similar to (Lee et al., 2009) and (Zeiler et al., 2010). In both of these studies a more powerful Pyramid Match Kernel SVM classifier (Lazebnik et al., 2006) is used to match the same level of performance. Figure 5.3 shows 128 filters that connect to 8 first layer features. Each row of filters connect a particular second layer feature map. It is seen that each row of filters extract similar features since their output response is summed together to form one output feature map.

**One Stage System:** We train 64 convolutional unsupervised features using both $\mathbf{SD}^{tanh}$ and $\mathbf{CD}^{shrink}$ methods. We use the encoder function obtained from this training followed by absolute value rectification, contrast normalization and

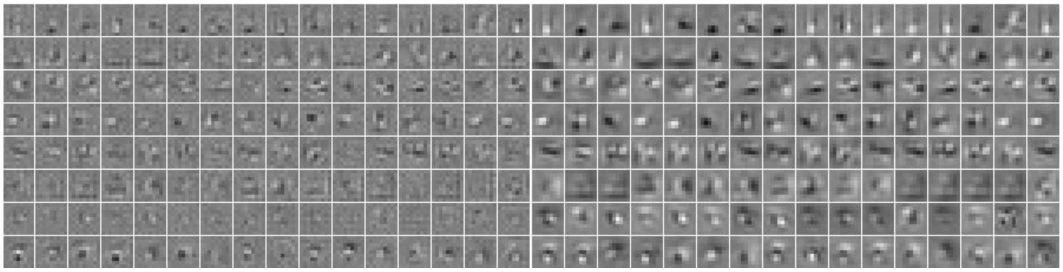

Figure 5.3: Second stage filters. **Left:** Encoder kernels that correspond to the dictionary elements. **Right:** 128 dictionary elements, each row shows 16 dictionary elements, connecting to a single second layer feature map. It can be seen that each group extracts similar type of features from their corresponding inputs.

| **Logistic Regression Classifier** | | | |
|---|---|---|---|
| | $\mathbf{SD}^{tanh}$ | $\mathbf{CD}^{shrink}$ | **PSD (Jarrett et al., 2009)** |
| **U** | $57.1 \pm 0.6\%$ | $57.3 \pm 0.5\%$ | 52.2% |
| $\mathbf{U}^{+}$ | $57.6 \pm 0.4\%$ | $56.4 \pm 0.5\%$ | 54.2% |

Table 5.1: Comparing $\mathbf{SD}^{tanh}$ encoder to $\mathbf{CD}^{shrink}$ encoder on Caltech 101 dataset using a single stage architecture. Each system is trained using 64 convolutional filters. The recognition accuracy results shown are very similar for both systems.

average pooling. The convolutional filters used are $9 \times 9$. The average pooling is applied over a $10 \times 10$ area with 5 pixel stride. The output of first layer is then $64 \times 26 \times 26$ and fed into a logistic regression classifier and Lazebnik's PMK-SVM classifier (Lazebnik et al., 2006) (that is, the spatial pyramid pipeline is used, using our features to replace the SIFT features).

**Two Stage System:** We train 4096 convolutional filters with $\mathbf{SD}^{tanh}$ method using 64 input feature maps from first stage to produce 256 feature maps. The second layer features are also $9 \times 9$, producing $256 \times 18 \times 18$ features. After applying absolute value rectification, contrast normalization and average pooling (on a $6 \times 6$ area with stride 4), the output features are $256 \times 4 \times 4$ (4096) dimensional. We only use multinomial logistic regression classifier after the second layer feature extraction stage.

We denote unsupervised trained one stage systems with $U$, two stage unsupervised trained systems with $UU$ and "+" represents supervised training is performed afterwards. $R$ stands for randomly initialized systems with no unsupervised training.

Comparing our $U$ system using both $\mathbf{SD}^{tanh}$ and $\mathbf{CD}^{shrink}$ (57.1% and 57.3%) with the 52.2% reported in (Jarrett et al., 2009), we see that convolutional training results in significant improvement. With two layers of purely unsupervised features ($UU$, 65.3%), we even achieve the same performance as the patch-based model of Jarrett et al. (Jarrett et al., 2009) after supervised fine-tuning (63.7%). Moreover, with additional supervised fine-tuning ($U^+U^+$) we match or perform very close

| **Logistic Regression Classifier** | |
|---|---|
| **PSD (Jarrett et al., 2009)** (**UU**) | 63.7 |
| **PSD (Jarrett et al., 2009)** ($\mathbf{U}^+\mathbf{U}^+$) | 65.5 |
| **SD**$^{tanh}$ (**UU**) | 65.3 ± 0.9% |
| **SD**$^{tanh}$ ($\mathbf{U}^+\mathbf{U}^+$) | 66.3 ± 1.5% |
| **PMK-SVM (Lazebnik et al., 2006) Classifier:**<br>Hard quantization + multiscale pooling<br>+ intersection kernel SVM | |
| **SIFT (Lazebnik et al., 2006)** | 64.6 ± 0.7% |
| **RBM (Lee et al., 2009)** | 66.4 ± 0.5% |
| **DN (Zeiler et al., 2010)** | 66.9 ± 1.1% |
| **SD**$^{tanh}$ (**U**) | 65.7 ± 0.7% |

Table 5.2: Recognition accuracy on Caltech 101 dataset using a variety of different feature representations using two stage systems and two different classifiers.

to (66.3%) similar models (Lee et al., 2009; Zeiler et al., 2010) with two layers of convolutional feature extraction, even though these models use the more complex spatial pyramid classifier (PMK-SVM) instead of the logistic regression we have used; the spatial pyramid framework comprises a codeword extraction step and an SVM, thus effectively adding one layer to the system. We get 65.7% with a spatial pyramid on top of our single-layer $U$ system (with 256 codewords jointly encoding $2 \times 2$ neighborhoods of our features by hard quantization, then max pooling in each cell of the pyramid, with a linear SVM, as proposed by authors in (Boureau et al., 2010)).

Our experiments have shown that sparse features achieve superior recognition performance compared to features obtained using a dictionary trained by a patch-based procedure as shown in Table 5.1. It is interesting to note that the improvement is larger when using feature extractors trained in a purely unsupervised way, than when unsupervised training is followed by a supervised training phase (57.1 to 57.6). Recalling that the supervised tuning is a *convolutional* procedure, this last training step might have the additional benefit of decreasing the redundancy between patch-based dictionary elements. On the other hand, this contribution would be minor for dictionaries which have already been trained convolutionally in the unsupervised stage.

### 5.2.2 Pedestrian Detection

We train and evaluate our architecture on the INRIA Pedestrian dataset (Dalal and Triggs, 2005) which contains 2416 positive examples (after mirroring) and 1218 negative full images. For training, we also augment the positive set with small translations and scale variations to learn invariance to small transformations, yielding 11370 and 1000 positive examples for training and validation respectively. The negative set is obtained by sampling patches from negative full images at random scales and locations. Additionally, we include samples from the positive set with larger and smaller scales to avoid false positives from very different scales. With these additions, the negative set is composed of 9001 training and 1000 validation samples.

#### 5.2.2.1 Architecture and Training

A similar architecture as in the previous section was used, with 32 filters, each $7 \times 7$ for the first layer and 64 filters, also $7 \times 7$ for the second layer. We used $2 \times 2$ average pooling between each layer. A fully connected linear layer with 2 output scores (for pedestrian and background) was used as the classifier. We trained this system on $78 \times 38$ inputs where pedestrians are approximately 60 pixels high. We have trained our system with and without unsupervised initialization, followed by fine-tuning of the entire architecture in supervised manner. Figure 5.5 shows comparisons of our system with other methods as well as the effect of unsupervised initialization.

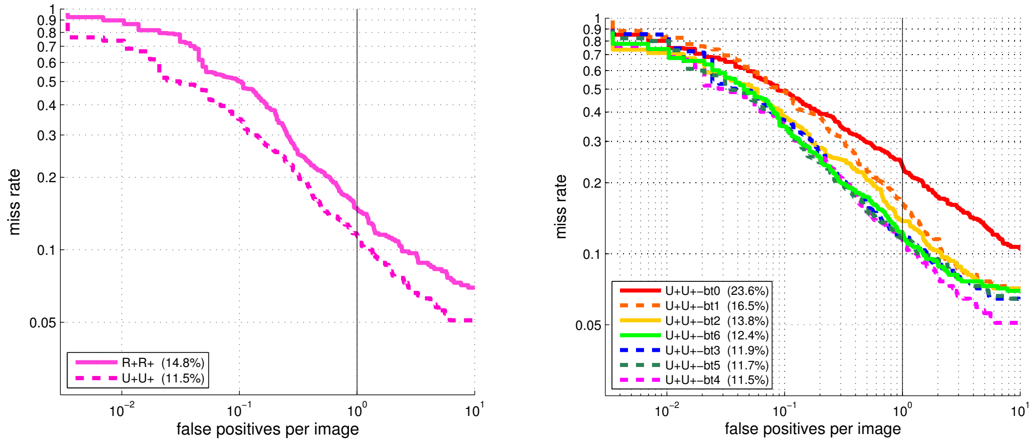

Figure 5.4: Results on the INRIA dataset with per-image metric. **Left:** Comparing two best systems with unsupervised initialization ($UU$) vs random initialization ($RR$). **Right:** Effect of bootstrapping on final performance for unsupervised initialized system.

After one pass of unsupervised and/or supervised training, several bootstrapping passes were performed to augment the negative set with the 10 most offending samples on each full negative image and the bigger/smaller scaled positives. We select the most offending sample that has the biggest opposite score. We limit the number of extracted false positives to 3000 per bootstrapping pass. As (Walk et al., 2010) showed, the number of bootstrapping passes matters more than the initial training set. We find that the best results were obtained after four passes, as shown in figure 5.5 improving from 23.6% to 11.5%.

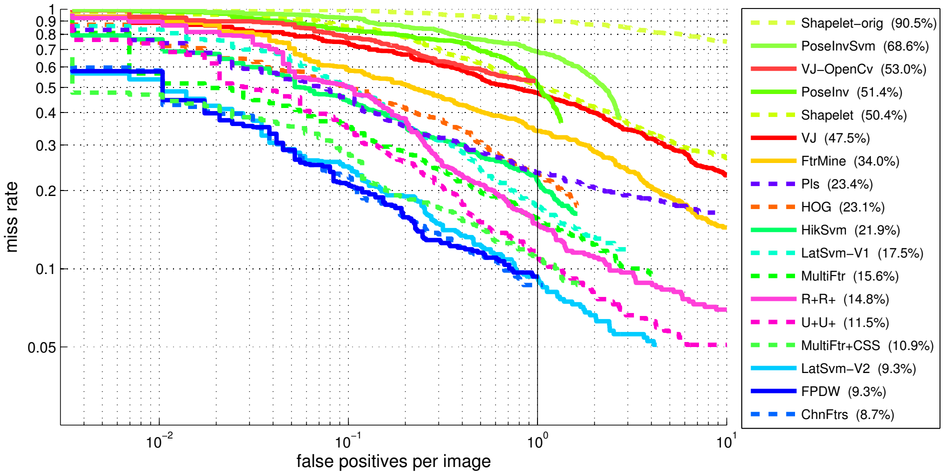

Figure 5.5: Results on the INRIA dataset with per-image metric. These curves are computed from the bounding boxes and confidences made available by (Dollár et al., 2009b). Comparing our two best systems labeled ($U^{+}U^{+}$ and $R^{+}R^{+}$)with all the other methods.

#### 5.2.2.2 Per-Image Evaluation

Performance on the INRIA set is usually reported with the per-window methodology to avoid post-processing biases, assuming that better per-window performance yields better per-image performance. However (Dollár et al., 2009b) empirically showed that the per-window methodology fails to predict the performance per-image and therefore is not adequate for real applications. Thus, we evaluate the per-image accuracy using the source code available from (Dollár et al., 2009b), which matches bounding boxes with the 50% PASCAL matching measure ($\frac{intersection}{union} > 0.5$).

In figure 5.5, we compare our best results (11.5%) to the latest state-of-the-art results (8.7%) gathered and published on the Caltech Pedestrians website[1]. The results are ordered by miss rate (the lower the better) at 1 false positive per image on average (1 FPPI). The value of 1 FPPI is meaningful for pedestrian detection because in real world applications, it is desirable to limit the number of false alarms.

It can be seen from figure 5.4 that unsupervised initialization significantly improves the performance (14.8% vs 11.5%). The number of labeled images in INRIA dataset is relatively small, which limits the capability of supervised learning algorithms. However, an unsupervised method can model large variations in pedestrian pose, scale and clutter with much better success.

Top performing methods (Dollár et al., 2009a), (Dollár et al., 2010), (Felzen-

[1] http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/files/data-INRIA

szwalb et al., 2010), (Walk et al., 2010) also contain several components that our simplistic model does not contain. Probably, the most important of all is color information, whereas we have trained our systems only on gray-scale images. Another important aspect is training on multi-resolution inputs (Dollár et al., 2009a), (Dollár et al., 2010), (Felzenszwalb et al., 2010). Currently, we train our systems on fixed scale inputs with very small variation. Additionally, we have used much lower resolution images than top performing systems to train our models ($78 \times 38$ vs $128 \times 64$ in (Walk et al., 2010)). Finally, some models (Felzenszwalb et al., 2010) use deformable body parts models to improve their performance, whereas we rely on a much simpler pipeline of feature extraction and linear classification.

Our aim in this work was to show that an adaptable feature extraction system that learns its parameters from available data can perform comparably to best systems for pedestrian detection. We believe by including color features and using multi-resolution input our system's performance would increase.

# Chapter 6

# Sparse Coding by Variational Marginalization

In chapters 2 and 3, we have shown that the optimal solution for the sparse coding problem is unstable under small perturbations of the input signal. The predictor function $\mathcal{F}_e$ provides a smooth mapping, and alleviates the stability problem as shown in chapter 2. Additionally we have developed an extension to learn invariance from data in chapter 3 by extending the sparse coding problem using complex cell models from neuroscience which is equivalent to the group sparse coding formulation with overlapping pools. However, we have always assumed that the optimal solution is the minimum of a certain energy function, amounting to finding the maximum a posteriori (MAP) solution, rather than the maximum likelihood solution. In this chapter, we provide a variational marginalization (MacKay, 2003) model of the sparse coding problem for finding more stable representations. Very similar approaches that use variational formulation for sparse modeling have already been published in the literature (Girolami, 2001; Wipf et al., 2004; Seeger, 2008). In these works, both the code inference and dictionary update steps are modeled using variational marginalization, however, we only formulate the sparse coding using variational methods and keep using stochastic gradient descent for dictionary updates since the instability of the solution is due to the MAP solution of code inference using an overcomplete dictionary.

## 6.1 Variational Marginalization for Sparse Coding

One can write the probability distribution corresponding to a given energy function through Gibbs distribution as:

$$\begin{aligned} P(x;\theta) =& \frac{e^{-\beta E(x;\theta)}}{Z(\beta,\theta)} \\ Z(\beta,\theta) =& \int_x e^{-\beta E(x;\theta)} \end{aligned} \tag{6.1}$$

where $Z(\beta,\theta)$ is the partition function (normalization coefficient). For high dimensional problems, computing $Z$ is intractable and Energy-Based Models (LeCun et al., 2006) avoid computing this term. On the other hand, using variational free energy minimization techniques (MacKay, 2003), one can find a simpler distribution $Q(x;\gamma)$ that approximates $P(x)$ and find a lower bound to $Z(\beta,\theta)$.

The free energy of a system is:

$$F_\beta = -\frac{1}{\beta}\log\int_x e^{-\beta E(x;\theta)} \tag{6.2}$$

and the corresponding variational free energy under the approximating distribution $Q(x;\gamma)$ is:

$$\begin{aligned}
\beta\tilde{F_\beta} &= \int_x Q(x;\gamma)\log\frac{Q(x;\gamma)}{e^{-\beta E(x;\theta)}} && (6.3)\\
\tilde{F_\beta} &= \mathbb{E}_Q\left[E(x;\theta)\right] - \frac{1}{\beta}H(Q) && (6.4)
\end{aligned}$$

by substituting the probability distribution from equation (6.1):

$$\beta\tilde{F_\beta} = \int_x Q(x;\gamma)\log\frac{Q(x;\gamma)}{P(x;\theta)} - \log Z(\beta,\theta) \tag{6.5}$$

$$\beta\tilde{F_\beta} = D_{KL}(Q||P) + \beta F \tag{6.6}$$

one can see that the variational free energy $\tilde{F}$ is an upper bound on the free energy and the difference is defined by the KL-divergence between $Q$ and $P$.

In this section we focus on the $\ell^1$ minimization sparse coding model without the predictor term:

$$E_{SC}(X,z,D) = \min\frac{1}{2}||x - Dz||_2^2 + \lambda\sum_i |z_i| \tag{6.7}$$

and rewriting eq. (6.4), the corresponding variational free energy is:

$$F_\beta(\tilde{X};D) = \int_z Q(z)E(X,z;D) - \frac{1}{\beta}H(Q(z)) \tag{6.8}$$

The above equation is valid for any distribution $Q(x)$ and we assume a Gaussian with mean $m$ and diagonal variance $S$ (s.t $\sigma_i^2 = s_i = S_{ii}, i = 1..d$)

Below we give a detailed expansion of equation (6.8). Expectation of energy term can be expanded as follows:

$$
\begin{aligned}
&\int_z Q(z)E(X,z;D) \\
&= \int_z Q(z)||X - Dz||_2^2 + \lambda \int_z Q(z)||z||_1 && (6.9)\\
&= \int_z Q(z)(X - Dz)^T(X - Dz) + \lambda \int_z Q(z)||z||_1 && (6.10)\\
&= \int_z Q(z)\left[X^T X - 2X^t Dz + z^T D^T Dz\right] + \lambda \int_z Q(z)||z||_1 && (6.11)
\end{aligned}
$$

By using linearity of expectation and the fact that $z^T Az = \mathbf{Tr}\left[Azz^T\right]$, the quadratic term can be simplified as:

$$
\begin{aligned}
&= X^T X - 2X^T Dm + m^T D^T Dm + \mathbf{Tr}\left[D^T DS\right] + \lambda \int_z Q(z)||z||_1 && (6.12)\\
&= ||X - Dm||_2^2 + \mathbf{Tr}\left[D^T DS\right] + \lambda \sum_i \int_{z_i} Q(z_i)\,|z_i| && (6.13)
\end{aligned}
$$

A random variable $(y)$ such that $y = |x|$ and $x \in \mathcal{N}(\mu, \sigma^2)$ has folded normal distribution (Leone et al., 1961). The pdf of $y$ is given as:

$$
f(y) = \frac{1}{\sigma\sqrt{2\pi}}\left[e^{-(x-\mu)^2/2\sigma^2} + e^{-(x+\mu)^2/2\sigma^2}\right] \tag{6.14}
$$

The expected value of $y$ is calculated by integration:

$$
\mathbb{E}[y] = \frac{1}{\sigma\sqrt{2\pi}}\int_0^\infty \left[e^{-(x-\mu)^2/2\sigma^2} + e^{-(x+\mu)^2/2\sigma^2}\right] \tag{6.15}
$$

The first term can be solved by change of variables. Let $t = \frac{x-\mu}{\sigma\sqrt{2}}$:

$$= \sigma\sqrt{\frac{2}{\pi}} \int\limits_{-\mu/\sigma\sqrt{2}}^{\infty} e^{-t^2} t dt + \frac{\mu}{\sqrt{\pi}} \int\limits_{-\mu/\sigma\sqrt{2}}^{\infty} e^{-t^2} dt \tag{6.16}$$

$$= \sigma\sqrt{\frac{2}{\pi}} \left[ -\frac{1}{2} e^{-t^2} \Big|_{-\mu/\sigma\sqrt{2}}^{\infty} \right] + \frac{\mu}{\sqrt{\pi}} \int\limits_{0}^{\mu/\sigma\sqrt{2}} e^{-t^2} dt + \frac{\mu}{\sqrt{\pi}} \int\limits_{0}^{\infty} e^{-t^2} dt \tag{6.17}$$

$$= \frac{\sigma}{\sqrt{2\pi}} e^{-\mu^2/2\sigma^2} + \frac{\mu}{2} \left[ 1 + erf(\mu/\sigma\sqrt{2}) \right] \tag{6.18}$$

The second term can also be solved by change of variables similarly. In this case, let $t = \frac{x+\mu}{\sigma\sqrt{2}}$:

$$= \sigma\sqrt{\frac{2}{\pi}} \int\limits_{\mu/\sigma\sqrt{2}}^{\infty} e^{-t^2} t dt - \frac{\mu}{\sqrt{\pi}} \int\limits_{\mu/\sigma\sqrt{2}}^{\infty} e^{-t^2} dt \tag{6.19}$$

$$= \sigma\sqrt{\frac{2}{\pi}} \left[ -\frac{1}{2} e^{-t^2} \Big|_{\mu/\sigma\sqrt{2}}^{\infty} \right] - \left[ \frac{\mu}{\sqrt{\pi}} \int\limits_{0}^{\infty} e^{-t^2} dt - \frac{\mu}{\sqrt{\pi}} \int\limits_{0}^{\mu/\sigma\sqrt{2}} e^{-t^2} dt \right] \tag{6.20}$$

$$= \frac{\sigma}{\sqrt{2\pi}} e^{-\mu^2/2\sigma^2} - \frac{\mu}{2} \left[ 1 - erf(\mu/\sigma\sqrt{2}) \right] \tag{6.21}$$

Therefore, for each $i$, the expected value of $|z_i|$ is given as:

$$\mathbb{E}\left[|z_i|\right] = \sqrt{\frac{2}{\pi}} \sigma_i e^{-m_i^2/2\sigma_i^2} + m_i \left[ erf(m_i/\sigma_i\sqrt{2}) \right] \tag{6.22}$$

where we have used $erf(a) = \frac{2}{\sqrt{\pi}} \int_0^a e^{-t^2} dt$. This result can also be stated in terms of the cdf of standard normal distribution ($\Phi(a)$):

$$
\begin{aligned}
\Phi(a) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{a} e^{-t^2/2} dt \\
\Phi(a) &= \frac{1}{2}\left[1 + erf(a/\sqrt{2})\right] \\
\mathbb{E}\left[|z_i|\right] &= \sqrt{\frac{2}{\pi}} \sigma_i e^{-m_i^2/2\sigma_i^2} + m_i \left[1 - 2\Phi(\frac{-m_i}{\sigma_i})\right] \qquad (6.23)
\end{aligned}
$$

Finally, the entropy of a gaussian distribution is given as:

$$
\begin{aligned}
H(Q(z)) &= \frac{1}{2}\ln[(2\pi e)^d |S|] \qquad (6.24) \\
&= \frac{1}{2}\ln[(2\pi e)^d \prod_i s_i] \qquad (6.25)
\end{aligned}
$$

Combining equations 6.13, 6.23, 6.25 into 6.8:

$$
\begin{aligned}
\tilde{F_\beta}(X; D) = ||X - Dm||_2^2 + \mathbf{Tr}\left[D^T D S\right] + \\
\lambda \sum_i \left[\sqrt{\frac{2}{\pi}} \sigma_i e^{-m_i^2/2\sigma_i^2} + m_i \left[1 - 2\Phi(\frac{-m_i}{\sigma_i})\right]\right] + \\
- \frac{1}{2\beta} \ln \left[(2\pi e)^d \prod_i s_i\right] \qquad (6.26)
\end{aligned}
$$

At this point, we can solve for mean $m_i$ and standard deviation $\sigma_i$ from the derivative of equation 6.26.

$$
\begin{aligned}
\frac{\partial \tilde{F_\beta}}{\partial m_i} &= \left[-2D^T(X - Dm)\right]_i + \\
&\lambda \left[\sqrt{\frac{2}{\pi}} \frac{m_i}{\sigma_i} e^{-m_i^2/2\sigma_i^2} + \left[1 - 2\Phi(-\frac{m_i}{\sigma_i})\right] + \sqrt{\frac{2}{\pi}} \frac{m_i}{\sigma_i} e^{-m_i^2/2\sigma_i^2}\right] \\
\frac{\partial \tilde{F_\beta}}{\partial m_i} &= \left[-2D^T(X - Dm)\right]_i + \lambda \left[1 - 2\Phi(-\frac{m_i}{\sigma_i})\right] \qquad (6.27)
\end{aligned}
$$

$$\begin{aligned}
\frac{\partial \tilde{F}_\beta}{\partial \sigma_i} &= diag(D^T D)_i + \\
&\quad \lambda \left[ \sqrt{\frac{2}{\pi}} e^{-m_i^2/2\sigma_i^2} + \sqrt{\frac{2}{\pi}} \frac{m_i^2}{\sigma_i^2} e^{-m_i^2/2\sigma_i^2} - \sqrt{\frac{2}{\pi}} \frac{m_i^2}{\sigma_i^2} e^{-m_i^2/2\sigma_i^2} \right] - \frac{1}{\beta \sigma_i} \\
\frac{\partial \tilde{F}_\beta}{\partial \sigma_i} &= diag(D^T D)_i + \lambda \sqrt{\frac{2}{\pi}} e^{-m_i^2/2\sigma_i^2} - \frac{1}{\beta \sigma_i} \qquad (6.28)
\end{aligned}$$

Using these equations, a standard CG solver can be used to alternatively optimize the mean and the standard deviation for each unit. The energy function can be separated into two parts: first computing the mean of the approximating distribution $Q$ and second, computing the standard deviation of $Q$. After convergence, the mean values ($m_i$) are used as the representation (code) instead of $z^*$ that is obtained from minimizing the $\ell^1$ energy. The relation between the $\ell^1$ energy minimization, given in eq. (6.7), and the variational marginalization in eq. (6.26) can be better understood from the following equation that only contains the variational energy terms that depend on the mean of $Q$. In figure 6.1, a comparison of the sparsity inducing term in variational minimization and the absolute value function is compared.

$$\tilde{F}_{\beta_{mean}} = ||X - Dm||_2^2 + g(m; \sigma) \qquad (6.29)$$

$$g(m; \sigma) = \lambda \sum_i \left[ \sqrt{\frac{2}{\pi}} \sigma_i e^{-m_i^2/2\sigma_i^2} + m_i \left[ erf\left( \frac{m_i}{\sigma_i \sqrt{2}} \right) \right] \right] \qquad (6.30)$$

It can be seen from figure 6.1 that, as the standard deviation decreases, the distribution becomes tighter around the mean, thus converges to energy minimization solution. However, since the standard deviation is inferred, the solution procedure

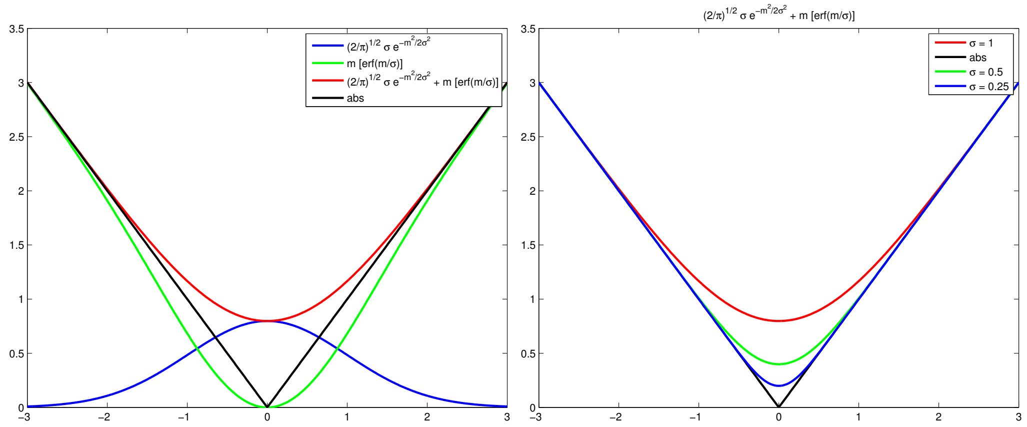

Figure 6.1: **Left:** The regularization term $g(m; \sigma)$ given in equation 6.29 is shown for $\sigma = 1$ together with absolute value function. **Right** The regularization term is shown for different values of $\sigma$.

chooses how closely it needs to approximate the energy minimization solution.

Having explained the variational formulation for sparse coding, we note that modeling the dictionary, i.e. updating the parameters of D, can also be formulated in a similar manner. However, in the following we continue using MAP solution for adopting D using stochastic gradient descent updates.

The proposed system can also be run convolutionally by using the derivative mask as explained in chapter 5. In figure 6.2, two convolutionally trained dictionaries are shown. On the left, a dictionary trained using sparse coding (Li and Osher) and on the right, a dictionary trained using the variational free energy minimization on MNIST handwritten digits dataset.

First we compare sparse coding and variational marginalization methods by

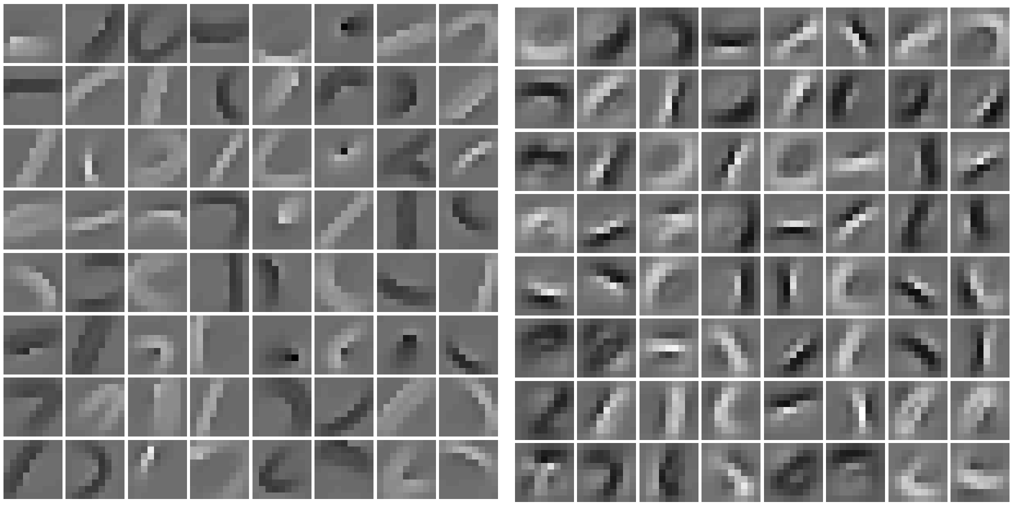

Figure 6.2: **Left:** Dictionary trained convolutionally using coordinate-descent sparse coding (Li and Osher) method on MNIST dataset. **Right:** Dictionary trained convolutionally using variational free energy minimization method on MNIST dataset.

training on natural images and comparing the reconstruction error vs sparsity behaviour. Figure 6.3 shows that the $\beta$ parameter controls the transition between energy minimization and variational free energy minimization. As it can be seen from eq. (6.8), as $\beta$ increases, the entropy term vanishes, thus variational marginalization solution approaches energy minimization solution.

## 6.2 Stability Experiments

Next, we compare the stability of representations obtained from sparse coding solution and variational free energy minimization. We take 60 consecutive frames from a video and plot the angle between representations for all consecutive pairs of frames as shown in figure 6.4. It can be seen that, as the sparsity coefficient $\lambda$ is decreased the sparse coding solution obtained using coordinate descent method (Li and Osher) produces less stable representations since the solution will contain many more active code units to achieve better reconstruction and using an overcomplete dictionary, there are many similar configurations to achieve same error level. The curves corresponding to variational free energy minimization are all produced using $\lambda = 0.5$ for varying values of $\beta$ and it can be seen that even for high values of $\beta$, variational free energy minimization produces more stable representations.

In order to quantify the effect of this stability improvement on recognition performance using variational free energy minimization, more controlled experiments on recognition and classification tasks should be performed.

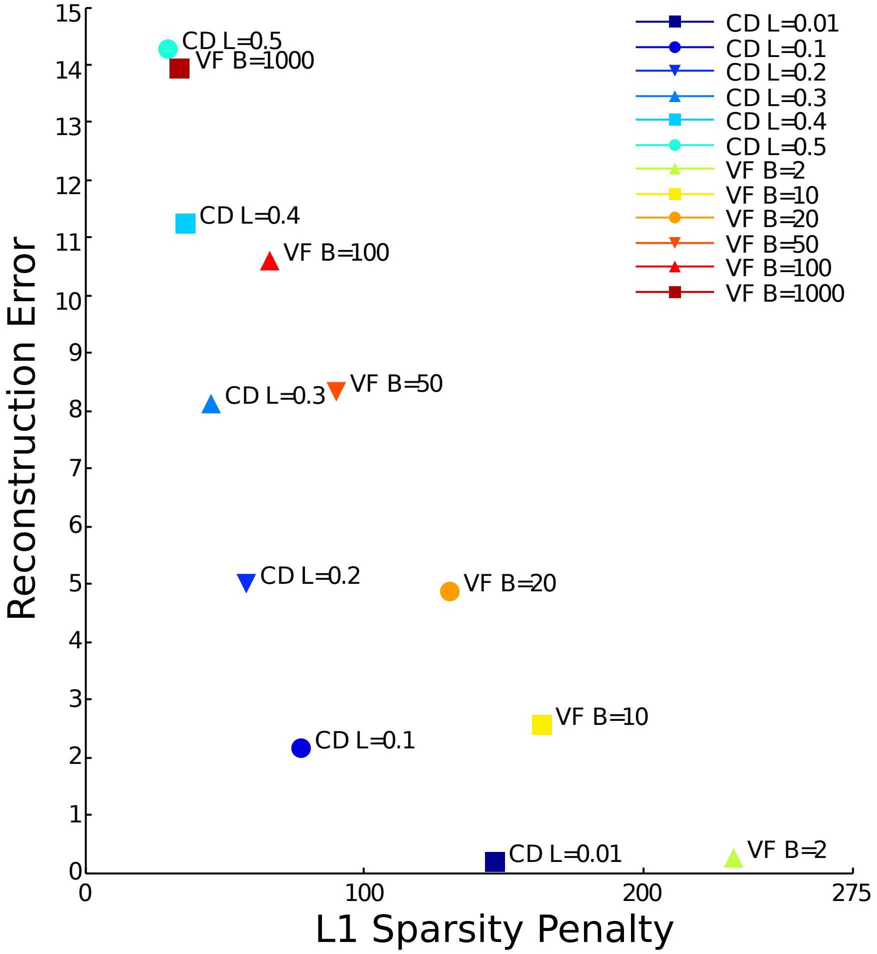

Figure 6.3: Reconstruction Error vs $\ell^1$ norm Sparsity Penalty for coordinate descent sparse coding and variational free energy minimization.

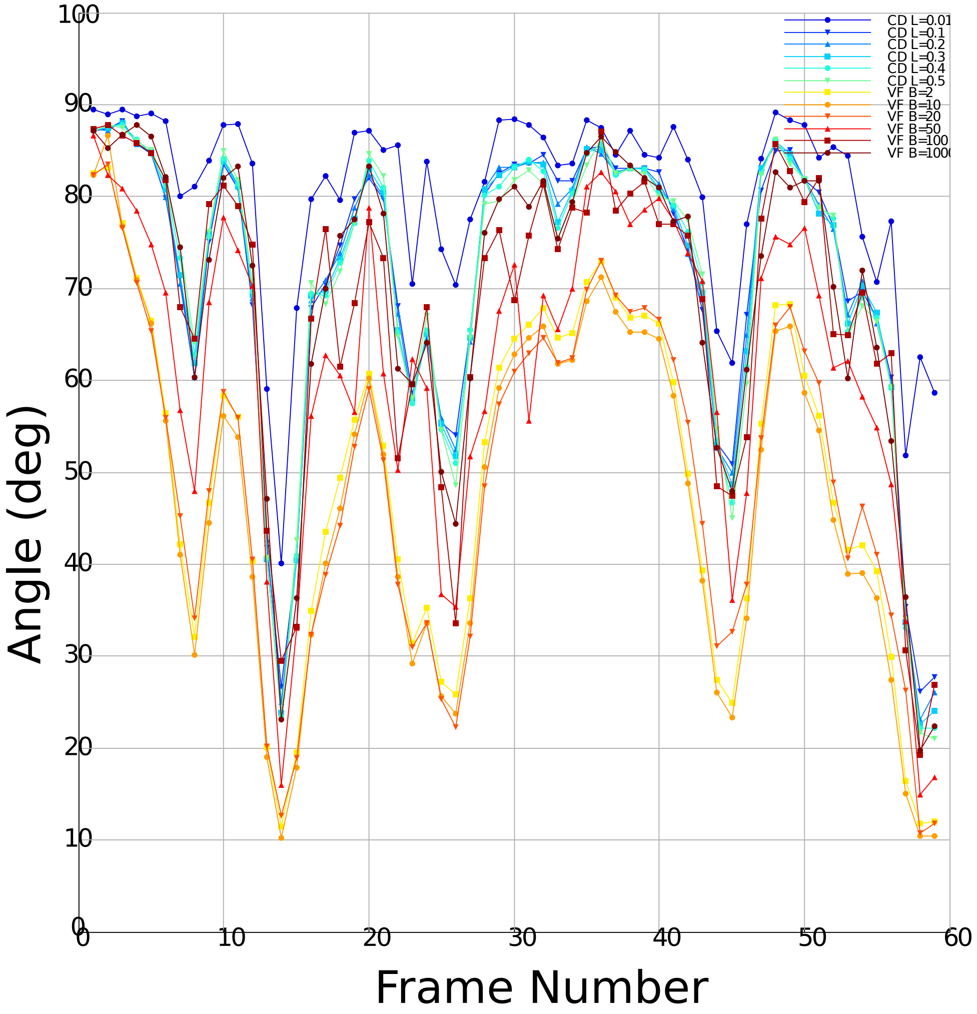

Figure 6.4: Angle between representations obtained for two consecutive frames using different parameter values using sparse coding and variational free energy minimization.

# Chapter 7

# Conclusion

In this thesis we have developed unsupervised methods for learning feature extractors that can be used in deep architectures for visual recognition. We have **1.** trained sparse models simultaneously with a feed-forward regressor to achieve efficient inference, **2.** merged complex cell models with sparse models to learn locally invariant feature extractors, **3.** developed convolutional sparse modeling that learns translation invariant dictionaries over images. **4.** shown that the non-linear functions that are used for building deep architectures have a very significant effect on the recognition performance.

In chapter 2, we have extended sparse modeling algorithms by jointly training a feed-forward predictor function. The resulting algorithm was named Predictive Sparse Decomposition (PSD). The output representations from this predictor function were then compared with optimal sparse representations on object recognition task using Caltech 101 dataset and slightly better performance was achieved. This modification enabled the use of sparse modeling algorithms for training generic feature extractors.

In chapter 3, we have shown that modeling invariant representations can be achieved using group sparsity criterion with overlapping group definitions instead of per-component $\ell^1$ sparsity criterion. When the trained feature extractor was compared with the SIFT feature extractor on the Caltech 101 dataset, it achieved very similar recognition performance, however, when applied on the MNIST handwritten digit classification task, the proposed system performed significantly better since the parameters are adapted to the particular task at hand, rather than being optimized for only natural images.

We have then used the PSD algorithm to build hierarchical feature extractors as explained in chapter 4 and tested on three benchmark datasets: Caltech 101, MNIST and NORB. We have shown that the form of the non-linear functions used in between stages of a hierarchical system become very important especially when there is lack of supervised labeled data.

Finally, in order to reduce the redundancy in the dictionary introduced from training on local image patches, we have developed convolutional sparse modeling. A dictionary that is trained convolutionally contains a richer set of features compared to the same size dictionary trained on image patches and and using the feature extractors trained convolutionally increases the recognition performance compared to patch based training.

The unsupervised methods proposed in this thesis are a continuation of a general unsupervised learning framework that contains a feed-forward predictor function and a linear decoding function which was first proposed in (Ranzato et al., 2006). (Ranzato et al., 2007b) extended the algorithm proposed in (Ranzato et al., 2006) to encode translational invariance and to build hierarchical feature extraction systems for object recognition. In (Ranzato et al., 2007a) a new sparsifying penalty function was introduced instead of the temporal soft-max used in (Ranzato et al., 2006) and the predictor and dictionary parameters were constrained to be transpose of each other to prevent them from growing. In PSD (Kavukcuoglu et al.), we used $\ell^1$ penalty for enforcing sparsity on the representation and also replaced the symmetry constraint with a unit L2-norm constraint on the dictionary elements, which also enabled us to use various different architectures in predictor function. We have then extended PSD using complex cell models to learn invariant representations in (Kavukcuoglu et al., 2009) and in (Kavukcuoglu et al., 2010) we introduced convolutional sparse modeling together with better predictor functions for sparse representations and efficient convolutional sparse coding algorithms.

In summary, we have developed algorithms and methods to achieve a successful object recognition system that has a hierarchical structure, can be trained using unlabeled data for improved accuracy, can produce locally invariant representations and finally efficient to be used on large datasets and real-time vision applications. There is much more to be done to achieve this goal, however we believe the models proposed in this work are valuable contributions towards realizing this objective. Several interesting problems are left open for future research. In particular, rather than using a fixed contrast normalization function, developing an adaptive function to reduce correlations between features, integrating invariant models into hierarchical structure and developing integrated learning algorithms for deep networks to improve on layer-wise training seem like the most important ones.

# Bibliography

M. Aharon, M. Elad, and A. M. Bruckstein. K-SVD and its non-negative variant for dictionary design. In M. Papadakis, A. F. Laine, and M. A. Unser, editors, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 5914 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 327–339, August 2005.

Michal Aharon, Michael Elad, and Alfred Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, November 2006.

Amr Ahmed, Kai Yu, Wei Xu, Yihong Gong, and Eric Xing. Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks. In *ECCV '08: Proceedings of the 10th European Conference on Computer Vision*, pages 69–82, Berlin, Heidelberg, 2008. Springer-Verlag.

Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Img. Sci.*, 2(1):183–202, 2009.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In Bernhard Schölkopf, John Platt, and Thomas Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007.

Y-Lan Boureau, Francis Bach, Yann LeCun, and Jean Ponce. Learning mid-level features for recognition. In *CVPR*, 2010.

Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1998.

R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning, ICML*, 2008.

P. Comon. Independent Component Analysis, a new concept ? *Signal Processing, Elsevier*, 36(3):287–314, April 1994. ISSN 0165-1684. doi: 10.1016/0165-1684(94)90029-9. Special issue on Higher-Order Statistics. hal-00417283.

Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In Cordelia Schmid, Stefano Soatto, and Carlo Tomasi, editors, *International Conference on Computer Vision & Pattern Recognition*, volume 2, pages 886–893, INRIA Rhône-Alpes, ZIRST-655, av. de l'Europe, Montbonnot-38334, June 2005.

I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 57(11):1413–1457, 2004.

P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *BMVC 2009, London, England.*, 2009a.

P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: A benchmark. In *CVPR*, June 2009b.

P. Dollár, S. Belongie, and P. Perona. The fastest pedestrian detector in the west. In *BMVC 2010, Aberystwyth, UK.*, 2010.

DL Donoho and M Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via $\ell^1$ minimization. *Proc Natl Acad Sci U S A*, 100(5):2197–2202, 2003.

Bradley Efron, Trevor Hastie, Lain Johnstone, and Robert Tibshirani. Least angle regression. *Annals of Statistics*, 32:407–499, 2004.

L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories. In *Workshop on Generative-Model Based Vision*, 2004.

P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. In *PAMI 2010*, 2010.

M. Girolami. A variational method for learning sparse and overcomplete representations. *Neural Computation*, 13(11):2517–2532, 2001.

I.J. Goodfellow, Q.V. Le, A.M. Saxe, H. Lee, and A.Y. Ng. Measuring invariances in deep networks. *Advances in neural information processing systems*, 22:646–654, 2009.

K. Grauman and T. Darrell. The pyramid match kernel: discriminative classification with sets of image features. *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, 2:1458–1465 Vol. 2, Oct. 2005.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proc. International Conference on Machine learning (ICML'10)*, 2010.

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

Fu-Jie Huang and Yann LeCun. Large-scale learning with svm and convolutional nets for generic object categorization. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'06)*. IEEE Press, 2006.

A Hyvarinen and P Hoyer. Emergence of phase- and shift-invariant features by decomposition of natural images into independent feature subspaces. *Neural Computation*, 12(7):1705–1720, 2000 Jul.

Aapo Hyvärinen and Patrik O. Hoyer. A two-layer sparse coding model learns simple and complex cell receptive fields and topography from natural images. *Vision Research*, 41(18):2413 – 2423, 2001.

Aapo Hyvarinen and Urs Koster. Complex cell pooling and the statistics of natural images. *Network*, 18(2):81–100, 2007.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV'09)*. IEEE, 2009.

Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. Technical report, Computational and Biological Learning Lab, Courant Institute, NYU. http://arxiv.org/abs/1010.3467.

Koray Kavukcuoglu, Marc'Aurelio Ranzato, Rob Fergus, and Yann LeCun. Learning invariant features through topographic filter maps. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR'09)*. IEEE, 2009.

Koray Kavukcuoglu, Pierre Sermanet, Y-Lan Boureau, Karol Gregor, Michaël Mathieu, and Yann LeCun. Learning convolutional feature hierachies for visual recognition. In *Advances in Neural Information Processing Systems (NIPS 2010)*, 2010.

Teuvo Kohonen. Emergence of invariant-feature detectors in the adaptive-subspace self-organizing map. *Biological Cybernetics*, 75(4):281–291, 1996.

H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, 1:1–40, 2009.

S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, 2:2169–2178, 2006.

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 1998a.

Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and Muller K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998b.

Yann LeCun and Corinna Cortes. Mnist dataset, 1998. http://yann.lecun.com/exdb/mnist/.

Yann LeCun, Fu-Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of CVPR'04*. IEEE Press, 2004.

Yann LeCun, Sumit Chopra, Raia Hadsell, Ranzato Marc'Aurelio, and Fu-Jie Huang. A tutorial on energy-based learning. In G. Bakir, T. Hofman, B. Schölkopf, A. Smola, and B. Taskar, editors, *Predicting Structured Data*. MIT Press, 2006.

H. Lee, R. Grosse, R. Ranganath, and A.Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 801–808. MIT Press, Cambridge, MA, 2007.

FC Leone, LS Nelson, and RB Nottingham. The folded normal distribution. *Technometrics*, pages 543–550, 1961.

Y. Li and S. Osher. Coordinate Descent Optimization for l1 Minimization with Application to Compressed Sensing; a Greedy Algorithm. *CAM Report*, pages 09–17.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

S Lyu and E P Simoncelli. Nonlinear image representation using divisive normalization. In *Proc. Computer Vision and Pattern Recognition*, pages 1–8. IEEE Computer Society, Jun 23-28 2008.

David J.C. MacKay. *Information theory, inference, and learning algorithms.* Cambridge Univ Pr, 2003. ISBN 0521642981.

J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Discriminative learned dictionaries for local image analysis. *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, June 2008.

J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 689–696. ACM, 2009.

S Mallat and Z Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397:3415, 1993.

D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

Joseph F. Murray and Kenneth Kreutz-Delgado. Learning sparse overcomplete codes for images. *J. VLSI Signal Process. Syst.*, 45(1-2):97–110, 2006.

Jim Mutch and David G. Lowe. Multiclass object recognition with sparse, localized features. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 11–18, Washington, DC, USA, 2006. IEEE Computer Society.

Y. Nesterov. *Gradient methods for minimizing composite objective function.* CORE, 2007.

B A Olshausen and D J Field. Natural image statistics and efficient coding. *Network*, 7:333–339, 1996.

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.

B.A. Olshausen. Sparse coding of time-varying natural images. In *in Proc. of the Int. Conf. on Independent Component Analysis and Blind Source Separation*, 2000.

Simon Osindero, Max Welling, and Geoffrey E Hinton. Topographic product models applied to natural scene statistics. *Neural Comput*, 18(2):381–414, 2006.

Nicolas Pinto, David D Cox, and James J DiCarlo. Why is real-world visual object recognition hard? *PLoS Comput Biol*, 4(1):e27, 01 2008.

Yanjun Qi, Pavel P. Kuksa, Ronan Collobert, Kunihiko Sadamasa, Koray Kavukcuoglu, and Jason Weston. Semi-supervised sequence labeling with self-learned features. In *Proc. International Conference on Data Mining (ICDM'09)*. IEEE, 2009.

Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems 19*. Cambridge, MA, 2006.

Marc'Aurelio Ranzato, Y-Lan Boureau, and Yann LeCun. Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems (NIPS 2007)*, 2007a.

Marc'Aurelio Ranzato, Fu-Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press, 2007b.

C.J. Rozell, D.H Johnson, R.G. Baraniuk, and B.A. Olshausen. Sparse coding via thresholding and local competition in neural circuits. *Neural Computation*, 2008.

O Schwartz and E P Simoncelli. Natural signal statistics and sensory gain control. *Nature Neuroscience*, 4(8):819–825, August 2001.

M.W. Seeger. Bayesian inference and optimal design for the sparse linear model. *The Journal of Machine Learning Research*, 9:759–813, 2008.

Thomas Serre, Lior Wolf, and Tomaso Poggio. Object recognition with features inspired by visual cortex. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, pages 994–1000, Washington, DC, USA, 2005. IEEE Computer Society.

Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images:

A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(11):1958–1970, 2008.

M J Wainwright, O Schwartz, and E P Simoncelli. Natural image statistics and divisive normalization: Modeling nonlinearity and adaptation in cortical neurons. In R Rao, B Olshausen, and M Lewicki, editors, *Probabilistic Models of the Brain: Perception and Neural Function*, chapter 10, pages 203–222. MIT Press, February 2002.

S. Walk, N. Majer, K. Schindler, and B. Schiele. New features and insights for pedestrian detection. In *CVPR 2010, San Francisco, California.*, 2010.

Max Welling, Geoffrey Hinton, and Simon Osindero. Learning sparse topographic representations with products of student-t distributions. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1359–1366. MIT Press, Cambridge, MA, 2003.

J. Weston, F. Rattle, and R. Collobert. Deep learning via semi-supervised embedding. In *International Conference on Machine Learning, ICML*, 2008.

D. Wipf, J. Palmer, and B. Rao. Perspectives on sparse Bayesian learning. In *Advances in Neural Information Processing Systems 16: Proceedings of the 2003 Conference*, page 249. The MIT Press, 2004.

Jianchao Yang, Kai Yu, Yihong Gong, and Thomas Huang. Linear spatial pyramid matching using sparse coding for image classification. In *CVPR*, 2009.

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society*, 68(1):49–67, 2006.

M.D. Zeiler, D. Krishnan, G.W. Taylor, and R. Fergus. Deconvolutional Networks. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'10)*, 2010.