Learning Hierarchical Feature Extractors For Image Recognition

Y-Lan Boureau

Learning Hierarchical Feature Extractors For Image Recognition

by

Y-Lan Boureau

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September 2012

_____

Yann LeCun

_____

Jean Ponce

DEDICATION

To my parents.

ACKNOWLEDGMENTS

I am above all grateful to Yann LeCun and Jean Ponce, for providing me patient and insightful guidance during my years as their student. Many thanks as well to the other members of my thesis committee for giving me feedback and ideas. I would like to thank Francis Bach for being such a great inspiration and sharp discussant; I have been very lucky to work with him, as well as Marc'Aurelio Ranzato, Nicolas Le Roux, Koray Kavukcuoglu, and Pierre Sermanet, and Samy Bengio and Jason Weston at Google. Many ideas in this thesis were born while discussing with members of the Willow and Sierra teams, and the Computational and Biological Learning Lab. Finally, I thank my family and friends for encouraging me and bearing with me during all these years.

This work was supported by NSF grant EFRI/COPN-0835878 to NYU, ONR contract N00014-09-1-0473 to NYU and by the European Research Council (VideoWorld and Sierra grants).

ABSTRACT

Telling cow from sheep is effortless for most animals, but requires much engineering for computers. In this thesis, we seek to tease out basic principles that underlie many recent advances in image recognition. First, we recast many methods into a common unsupervised feature extraction framework based on an alternation of *coding steps*, which encode the input by comparing it with a collection of reference patterns, and *pooling steps*, which compute an aggregation statistic summarizing the codes within some region of interest of the image. Within that framework, we conduct extensive comparative evaluations of many coding or pooling operators proposed in the literature. Our results demonstrate a robust superiority of sparse coding (which decomposes an input as a linear combination of a few visual words) and max pooling (which summarizes a set of inputs by their maximum value). We also propose *macrofeatures*, which import into the popular spatial pyramid framework the joint encoding of nearby features commonly practiced in neural networks, and obtain significantly improved image recognition performance. Next, we analyze the *statistical properties of max pooling* that underlie its better performance, through a simple theoretical model of feature activation. We then present results of experiments that confirm many predictions of the model. Beyond the pooling operator itself, an important parameter is the set of pools over which the summary statistic is computed. We propose *locality in feature configuration space* as a natural criterion for devising better pools. Finally, we propose ways to make coding faster and more powerful through *fast convolutional feedforward architectures*, and examine how to in-

corporate supervision into feature extraction schemes. Overall, our experiments offer insights into what makes current systems work so well, and state-of-the-art results on several image recognition benchmarks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

What's in a face? Judging by kids' drawings, two dots for the eyes and one line for the mouth, sometimes one more dot for the nose. In fact, one of the most popular systems for face detection, that of Viola and Jones (Viola and Jones, 2004), uses Haar filters, which are very simple patterns of black and white rectangles. Face detection now works very well and in real time. Unfortunately, not everything is as simple. What's in a rose? Or a cat? Or any of the thousands of categories for which we have a generic name? People and animals are very good at recognizing scenes and objects, but we see without really knowing what it is that we see, and admire artists who do and can capture the essence of a scene with a few well-chosen strokes.

0.1 Building an artificial vision system

Images are a collection of activations tied to a location, e.g., luminance value at different pixels for a digital image, or retina activations in biological vision. These representations do not lend themselves well to semantic interpretation. Instead, the pixel activations have to be recombined into features which are more amenable to further analysis.

The craftsmanship involved in extracting these features has often progressed by trial and error, and involves a lot of hand-coding. The engineering strategy is to rely on domain knowledge and intuition, however imperfect. This is successful to some extent; but the vast number of tasks to solve each requires their own specific solution, so this approach may not scale up (Bengio and LeCun, 2007). Another strategy is then to

turn to machine learning and devise suitable training criteria and optimization methods to learn parameters automatically. Machine learning algorithms still require a lot of engineering craft though: in particular, the *architecture* itself, which defines the set of candidate functions, has to be chosen. Practitioners may resort to imitation of a working model; e.g., animals have solved vision. Biologically-inspired models such as the HMAX model (Serre et al., 2005) or more recent V1-like models (Pinto et al., 2008) take this avenue. But knowing what the brain does is a hard task (Olshausen and Field, 2005), so is distinguishing the crucial from the anecdotal. Besides, airplanes were not invented by copying birds.

These approaches are all valid, and often end up converging to similar solutions, albeit with different motivations. For example, a sparse representation, i.e., a representation that uses only a few of many available basis functions, can be sought because it is has statistical independence properties (Olshausen and Field, 1997), allows good signal compression (Donoho, 2006), or yields more interpretable statistical models (Tibshirani et al., 1997). An $\ell_{1,2}$ group-sparsity penalty can be arrived at in an attempt to obtain topographical feature maps (Hyvärinen and Hoyer, 2001) or for statistical model selection (Yuan and Lin, 2006).

0.2 Goals

This thesis attempts to unify several successful image recognition systems by looking at the operations that they effectively implement, for whatever motivation. Our goal is threefold:

- Find common patterns in successful systems, and test how robust they are to new combinations,
- Get a theoretical understanding of why some strategies work, beyond mere empirical evidence,
- Leverage the gained insight to devise new models and provide useful guidelines for future architectures.

A guiding thread is the success of sparse coding in vision. Sparse coding has obtained very good results in many image applications such as image restoration (Mairal et al., 2009b), denoising (Elad and Aharon, 2006), recognition (Boureau et al., 2010a; Yang et al., 2009b), but it is still unclear what makes it work so well. Sparse coding in image recognition architectures is often used in conjunction with a *pooling* operation, that summarizes feature activation over an image region. A hypothesis to explain the good performance of sparse coding is that it protects information from destruction by the pooling step. Some characteristics of sparse coding may serve as starting point:

- Inputs are reconstructed as a linear combination of basis functions instead of just a copy of one basis function;
- Most basis functions are rarely active across inputs;
- Basis functions tend to be active for inputs that are similar to one another.

The first trait differentiates sparse coding from hard or soft vector quantization, which reconstructs an input as a single basis function. The second trait means that a pooling operation that preserves more information for rare features will work well with sparse

coding — and hard vector quantization as well. The third trait has as a consequence that any operation that combines nonzero values of single basis functions will have similar effects as if it had been applied *conditioned on the inputs being similar*. Thus, sparse coding induces an implicit context dependency. The second and third traits are shared with hard vector quantization.

In this thesis, we suggest in Chapter 3 that the first trait makes sparse coding generally more suited to representing images than vector quantization. We propose in Chapter 4 that the second trait may explain the excellent performance of max pooling with sparse coding and hard vector quantization, as observed in Chapter 3. The third trait motivates experiments in Chapter 5 which show improved performance when conditioning pooling on context (i.e., activations of a basis function at two locations are pooled together only if the context given by the activation of all basis functions are similar at both locations).

0.3 Contributions and organization of the thesis

The main contributions of this thesis can be summarized as follows:

- In Chapter 2, we recast much previous work into a common framework that uses an alternation of coding and pooling modules, and propose in Chapter 3 an extensive experimental evaluation of many combinations of modules within that framework. In addition to known modules, we propose a new supervised coding module in Chapter 7, and *macrofeatures* in Chapter 3 as a better way to combine one level of feature extraction to the next.

- One of the striking conclusions of that evaluation is that max pooling is robustly superior to average pooling in combination with many modules. We propose explanations for that superiority of max pooling in Chapter 4. We devise a simple theoretical model of feature activations and test its predictions with new experiments.

- We demonstrate in Chapter 5 that preserving locality in configuration space during pooling is an important ingredient of the good performance of many new recent algorithms.

- By combining these insights, we obtain state-of-the-art results on several image recognition benchmark.

- In Chapter 6, we propose several ways to make our architecture faster without losing too much performance. This is joint work with Marc'Aurelio Ranzato, Koray Kavukcuoglu, Pierre Sermanet, Karol Gregor and Michaël Mathieu.

The thesis is organized as follows. Chapter 1 gives some background on feature extraction. Chapter 2 proposes a common framework that unifies many feature extraction schemes. Chapter 3 conducts extensive evaluations of multiple combinations of feature extraction operators that have appeared in the literature and proposes *macrofeatures*, that represent neighboring feature activations jointly. Chapter 4 proposes and tests theoretical justifications for the success of max pooling. Chapter 5 looks at the influence of locality in configuration space when performing pooling. Chapter 6 examines ways to makes sparse coding faster and convolutional. Chapter 7 introduces supervision into architecture training. We then propose future work directions, and conclude. Most of

the work presented in this thesis has been published in (Boureau et al., 2010a; Boureau et al., 2010b; Boureau et al., 2011; Kavukcuoglu et al., 2010; Ranzato et al., 2007a; Ranzato et al., 2007b; Ranzato et al., 2007c).

1

RELATED WORK

This chapter proposes a brief overview of related work in image recognition.

1.1 Hand-crafted feature extraction models

The appearance of an image patch can be described in terms of the responses of a filter bank, such as Gabor filters, wavelets, or steerable filters (Freeman et al., 1991; Simoncelli et al., 1998). More recent descriptors combine filter responses to an aggregating (or *pooling*) step. This makes them both discriminative and robust to common perturbations of the input such as small translations. The scale-invariant feature transform (SIFT) (Lowe, 2004) and Histograms of Gradients (HOG) (Dalal and Triggs, 2005) descriptors use this strategy. In these methods, the dominant gradient orientations are measured in a number of regions, and are pooled over a neighborhood, resulting in a local histogram of orientations.

Bag-of-words methods have been successful in the field of text processing. In image applications (Sivic and Zisserman, 2003), local image patches are usually assigned an index in a codebook obtained without supervision, yielding representations that are 1) all-or-none, 2) extremely sparse, and 3) purely based on appearance. Bag-of-words classification consists of 1) extracting local features located densely or sparsely at interest points, 2) quantizing them as elements (codewords) from a dictionary (codebook), 3) accumulating codewords counts into normalized histograms, and 4) feeding the his-

tograms to a classifier.

Despite its coarseness (all spatial information is discarded), this method has proven surprisingly successful in visual object recognition tasks (Lazebnik et al., 2006). Refinements introduced at each step have resulted in state-of-the-art performance. This includes replacing generic features (e.g., Gabor functions, wavelets (Mallat, 1999), Laplacian filters) by the more powerful handcrafted features described above (e.g., SIFT (Lowe, 1999), HOG (Dalal and Triggs, 2005)), retaining some amount of spatial information by computing bags of words over cells of a coarse grid(Dalal and Triggs, 2005) or pyramid (Lazebnik et al., 2006) instead of the whole image, and using sophisticated kernels (e.g., histogram intersection, chi-squared, etc.) during classification (Lazebnik et al., 2006; Zhang et al., 2007).

Successful kernels such as the pyramid match kernel (Grauman and Darrell, 2005) and the histogram intersection kernel (Lazebnik et al., 2006) work by refining the measure of how well a region matches another one. However they still rely on the rather crude match of vector quantization to assign an index to each word.

The spatial pyramid (Lazebnik et al., 2006) has emerged as a popular framework to encapsulate more and more sophisticated feature extraction techniques (Boureau et al., 2010a; Gao et al., 2010; Wang et al., 2010; Yang et al., 2009b; Yang et al., 2010; Zhou et al., 2010). Many of the experiments described in this thesis are conducted within that framework.

1.2 Sparse coding

A code is *sparse* if most of its components are zero. Sparse modeling reconstructs an input as a linear combination of few basis functions. The coefficients used for reconstruction constitute the sparse code. While sparse coding has been around for a long time, it has become extremely popular in vision in recent years (e.g. (Mairal et al., 2009c; Raina et al., 2007; Yang et al., 2009b; Gao et al., 2010; Wang et al., 2010; Wright et al., 2009)).

Incorporating the sparse objective by directly penalizing the number of nonzero coefficients (i.e., using an ℓ_0 pseudo-norm penalty) leads to an NP-hard problem. To avoid this issue, greedy algorithms such as orthogonal matching pursuit (OMP) (Mallat and Zhang, 1993) can be used. Another option is to relax the ℓ_0 penalty into an ℓ_1 one, which makes the optimization tractable and induces sparsity (Donoho, 2006).

The problem of finding a sparse decomposition of a signal is often coupled with that of finding a suitable dictionary (or set of basis functions). Sparse coding can be performed on a dictionary composed of random input samples or a standard basis function such as a discrete cosine transform (DCT) basis (Ahmed et al., 1974) or other wavelets (Mallat, 1999), but decomposition over a learned dictionary has been shown to perform much better for reconstruction tasks (Elad and Aharon, 2006). Dictionary learning algorithms usually alternate between minimization over the code and over the dictionary (Olshausen and Field, 1997; Elad and Aharon, 2006; Mairal et al., 2009a).

Sparse coding has a proven track record in signal processing applications (Chen et al., 1999). Successful image applications include denoising (Elad and Aharon, 2006),

classification (Mairal et al., 2009c; Raina et al., 2007), face recognition (Wright et al., 2009), image super-resolution (Yang et al., 2008).

1.3 Trained deep architectures

Deep architectures extract representations with increasing levels of invariance and complexity (Goodfellow et al., 2009), well-suited to artificial intelligence tasks requiring highly non-linear features (Bengio, 2007). Training deep (multi-layer) architectures has long seemed impossible because backpropagation (LeCun and Bengio, 1995; LeCun et al., 1998b) of the gradient through the multiple layers was getting stuck in local minima (Tesauro, 1992). One notable exception is convolutional networks (LeCun et al., 1998a), which take advantage of the translation invariance of images by tying parameters across all locations of the image. They have enjoyed enduring success in handwritten digit and object recognition.

Early attempts were *supervised*, namely, they used labels (e.g., digits or object categories) to train the architecture by backpropagating the gradients of a supervised classifier making up the last layer of the architecture. Hinton et al. (Hinton and Salakhutdinov, 2006) have introduced a successful layer-by-layer *unsupervised* strategy to pretrain deep architectures. Unsupervised training learns a good model of the input that allows reconstruction or generation of input data; common applications include data compression, denoising, and recovery of corrupted data. In (Hinton and Salakhutdinov, 2006), the architecture is constructed as a stack of restricted Boltzmann machines (RBM) that are trained in sequence to model the distribution of inputs; the output of each RBM layer is the input of the next layer. The whole network is then trained (or "fine-tuned") with a

supervised algorithm. Supervised training of the whole network is successful when the weights are initialized by layer-wise unsupervised pretraining.

Restricted Boltzmann machines (Hinton, 2002; Hinton et al., 2006) minimize an approximation of the negative log likelihood of the data under the model. An RBM is a binary stochastic symmetric machine defined by an energy function of the form: $E(Y, Z) = -Z^T W^T Y - b_{enc}^T Z - b_{dec}^T Y$, where Y is the input and Z is the code giving the hidden units activation, W is the weight matrix and b_{enc} and b_{dec} give the biases of the hidden and visible units, respectively. This energy can be seen as a special case of the encoder-decoder architecture that pertains to binary data vectors and code vectors (Ranzato et al., 2007a). Training an RBM minimizes an approximation of the negative log likelihood loss function, averaged over the training set, through a gradient descent procedure. Instead of estimating the gradient of the log partition function, RBM training uses contrastive divergence (Hinton, 2002), which takes random samples drawn over a limited region around the training samples. Sparse (Lee et al., 2007) and convolutional (Lee et al., 2009) versions of RBMs have been proposed in the literature.

Sparse autoencoders (Ranzato et al., 2006; Ranzato et al., 2007c; Ranzato et al., 2007b; Kavukcuoglu et al., 2008; Jarrett et al., 2009) train a feedforward non-linear encoder to produce a fast approximation of the sparse code. The basic idea is to include three terms in the loss to minimize: (1) the error between the predicting code and the actual sparse code, (2) the error between the reconstruction obtained from the sparse code and the input, and (3) a sparsity penalty over the sparse code. Training alternates between minimization over the code and the encoder and decoder weights. When training has converged, the minimization to obtain the code becomes unnecessary and the pre-

dicted code is used directly. The models differ in how they induce sparsity (e.g., Ranzato et al.(Ranzato et al., 2006) use a specific sparsifying function, most other models of this type use an ℓ_1 penalty), how they prevent the weights from blowing up (symmetry of the encoder and decoder weights is imposed in (Ranzato et al., 2007b), the decoder weights are normalized in (Kavukcuoglu et al., 2008)), or what non-linearity they use — in fact, the main conclusion of (Jarrett et al., 2009) is that choosing non-linearity and pooling function is often more important than training the weights. These architectures have obtained good results in image denoising (Ranzato et al., 2007a), image and handwriten digit classification (Ranzato et al., 2007c; Ranzato et al., 2007b; Jarrett et al., 2009), although their performance is still slightly below that of systems that incorporate some handcrafted descriptors such as SIFT.

Other models used as building blocks of deep networks include semi-supervised embedding models (Weston et al., 2008; Collobert and Weston, 2008), denoising autoencoders (Vincent et al., 2008).

In practice, most deep models used in a realistic context have three or fewer layers. What makes them "deep", then, is a built-in recursive quality: the procedure of adding one layer could be repeated as many times as desired, producing potentially very deep architectures. The spatial pyramid model presented in Sec. (1.1) is generally not viewed as a deep architecture, being composed of heterogeneous modules (low-level descriptor extractor, hard vector quantization and pyramidal pooling, classification via an SVM). Nevertheless, the resulting architecture resembles many of the deep networks discussed in this section, as argued in the next chapter. The most salient difference is rather that *training* is replaced by hand-crafted feature extractors in the spatial pyramid.

1.4 Pooling

Many of the computer vision architectures presented so far comprise a *spatial pooling step*, which combines the responses of feature detectors obtained at nearby locations into some statistic that summarizes the joint distribution of the features over some region of interest. The idea of feature pooling originates in Hubel and Wiesel's seminal work on complex cells in the visual cortex (Hubel and Wiesel, 1962), and is related to Koenderink's concept of locally orderless images (Koenderink and Van Doorn, 1999). Pooling features over a local neighborhood creates invariance to small transformations of the input. The pooling operation is typically a sum, an average, a max, or more rarely some other commutative (i.e., independent of the order of the contributing features) combination rule.

Fukushima's neocognitron was an early biologically-inspired model that had layers of pooling units alternating with layers of coding units (Fukushima and Miyake, 1982). Other biologically-inspired models that use pooling include convolutional networks, which use average pooling (LeCun et al., 1990; LeCun et al., 1998a), or max pooling (Ranzato et al., 2007b; Jarrett et al., 2009), the HMAX class of models, which uses max pooling (Serre et al., 2005), and some models of the primary visual cortex area V1 (Pinto et al., 2008), which use average pooling. Many popular methods for feature extraction also use pooling, including SIFT (Lowe, 2004), histograms of oriented gradients (HOG) (Dalal and Triggs, 2005) and their variations.

2

A RECURSIVE FRAMEWORK FOR FEATURE EXTRACTION MODELS

Successful feature extraction algorithms (e.g., SIFT or HOG descriptors) are often used as black boxes within more complicated systems, which obscures the many similarities they share. In this chapter, we propose a unified framework for a generic feature extraction module, which accommodates many algorithms commonly used in modern vision systems; we analyze a popular implementation of SIFT descriptors as an example. This naturally leads to viewing image recognition architectures as stacks of feature extraction layers; this chapter presents results which suggest that adding more layers is not useful.

2.1 Common steps of feature extraction

Finding good image features is critical in modern approaches to category-level image classification. Many methods first extract low-level descriptors (e.g., Gabor filter responses, SIFT (Lowe, 2004) or HOG descriptors (Dalal and Triggs, 2005)) at interest point locations, or nodes in a dense grid. We consider the problem of combining these local features into a global image representation suited to recognition using a common classifier such as a support vector machine. Since global features built upon low-level ones typically remain close to image-level information without attempts at high-level, structured image description (in terms of parts for example), we will refer to them as

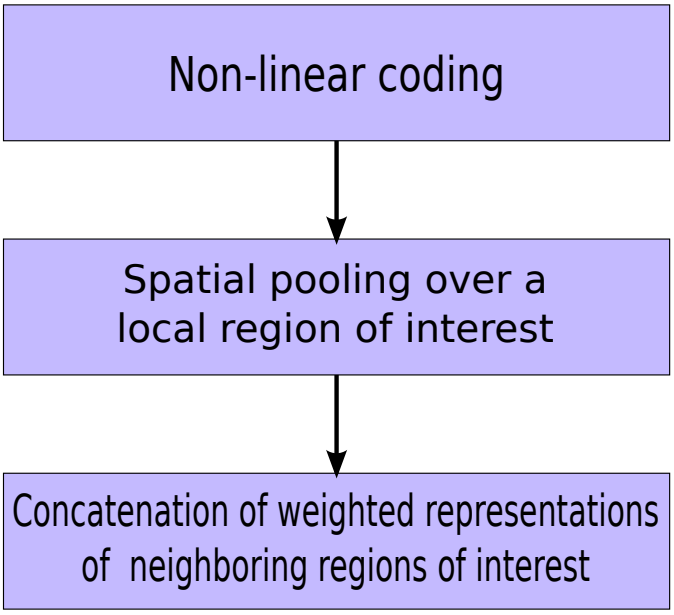
mid-level features.

Popular examples of mid-level features include bags of features (Sivic and Zisserman, 2003), spatial pyramids (Lazebnik et al., 2006), and the upper units of convolutional networks (LeCun et al., 1998a) or deep belief networks (Hinton and Salakhutdinov, 2006; Ranzato et al., 2007b). Extracting these mid-level features involves a sequence of interchangeable modules similar to that identified by Winder and Brown for local image descriptors (Winder and Brown, 2007). In this thesis, we focus on two types of modules:

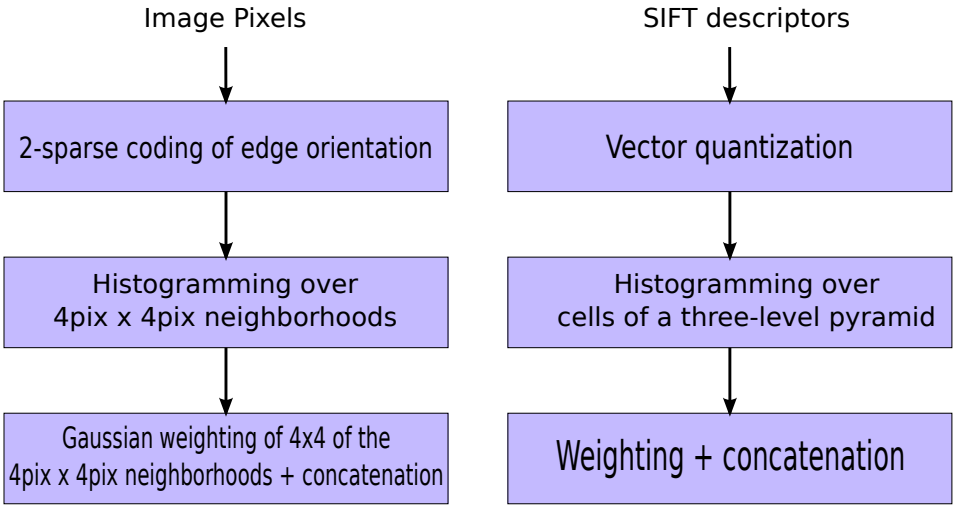
- **Coding:** Input features are locally transformed into representations that have some desirable properties such as compactness, sparseness (i.e., most components are 0), or statistical independence. The code is typically a vector with binary (vector quantization) or continuous (HOG, sparse coding) entries, obtained by decomposing the original feature on some codebook, or dictionary.

- **Spatial pooling:** The codes associated with local image features are pooled over some image neighborhood (e.g., the whole image for bags of features, a coarse grid of cells for the HOG approach to pedestrian detection, or a coarse hierarchy of cells for spatial pyramids). The codes within each cell are summarized by a single "semi-local" feature vector, common examples being the average of the codes (*average pooling*) or their maximum (*max pooling*).

Many low-level and mid-level feature extractors perform the same sequence of steps and are instances of a generic feature extractor (see Fig. (2.1)), which combines nonlinear coding with spatial pooling; these two steps are reminiscent of simple and complex cells in the mammalian visual cortex (see Fig. (2.2)). It could be argued that pooling



(a) Common steps of feature extraction



(b) SIFT descriptors

(c) Spatial Pyramid

Figure 2.1: Many mid-level features are extracted using the same sequence of modules as low-level descriptors. Top: generic feature extraction steps. Bottom, left: SIFT descriptor extraction. Bottom, right: mid-level feature extraction in the spatial pyramid.

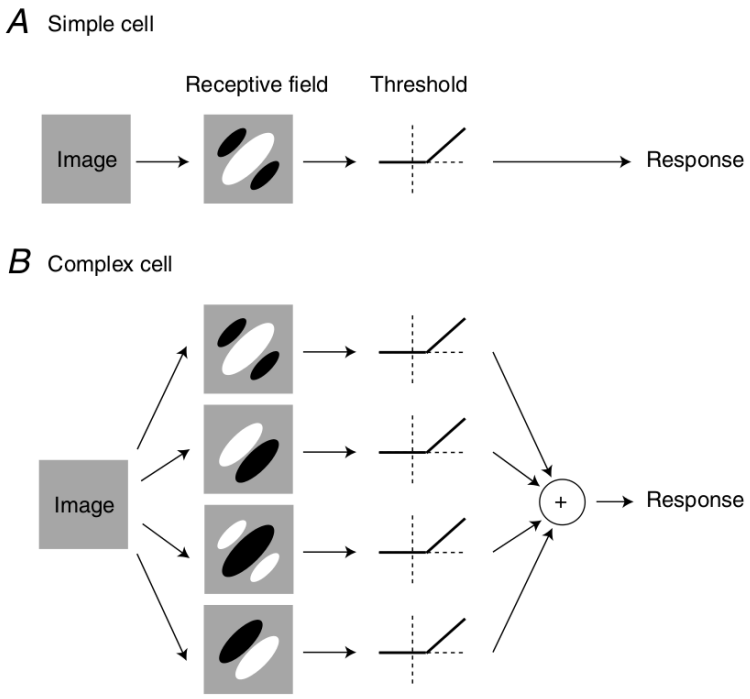


Figure 2.2: Standard model of the V1 area of the mammalian visual cortex. Figure from (Carandini, 2006).

is merely a special type of coding; what distinguishes them in this framework is that the coding step generally involves computing distances or dot-products of the input with a collection of stored vectors of same dimension, which have been called in the literature basis functions, templates, filters, codewords, atoms, centers, elements, or prototypes, while pooling computes orderless statistics over a neighborhood and discards some of the variation in the sample as irrelevant.

The output feature vector is formed by concatenating with suitable weights the vectors obtained for each pooling region, and can then be used as input to another layer of feature extraction.

The same coding and pooling modules can be plugged into various architectures. For example, average pooling is found in convolutional nets (LeCun et al., 1998a), bag-of-features methods, and HOG descriptors; max pooling is found in convolutional nets (Lee et al., 2009; Ranzato et al., 2007b), HMAX nets (Serre et al., 2005), and state-of-the-art variants of the spatial pyramid model (Yang et al., 2009b).

2.2 SIFT descriptors are sparse encoders of orientations

Lowe et al.'s SIFT descriptors (Lowe, 2004) have become ubiquitous in computer vision applications, and have indeed displayed superior performance in comparative studies (e.g., (Mikolajczyk and Schmid, 2005)). But they are often used as a black-box feature extractor, which may obscure the many common characteristics they share with most unsupervised feature extractors. In this section, we examine a popular implementation of SIFT, released as part of the spatial pyramid framework implementation, and show that it performs a smooth 2-dimensional sparse encoding of orientations over a set of

reference orientations.

We describe the default settings, used in the seminal spatial pyramid paper (Lazebnik et al., 2006): each 128-dimensional descriptor covers an area of 16×16 pixels, divided into a grid of 4×4 cells, and uses 8 reference edge orientations. SIFT descriptor extraction can be broken down into three steps: (1) sparse encoding into an 8-dimensional feature vector at every point, (2) local pooling and subsampling of the codes, and (3) concatenation and normalization.

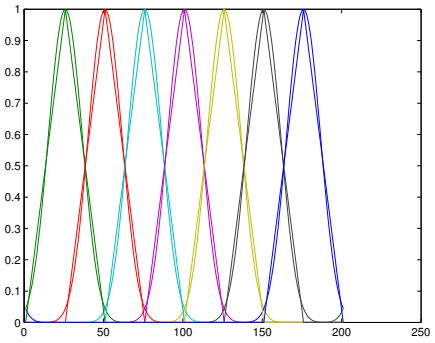
2.2.1 Sparse encoding of edge orientations

At each point, a continuous edge-orientation value is first obtained by computing the ratio of vertical and horizontal gradients, and extracting the corresponding oriented angle (in radians). The cosine of the difference between this angle and a set of $K = 8$ evenly spaced reference orientations is then computed, and the negative parts are truncated. The resulting values are then raised to the power $\gamma = 9$.

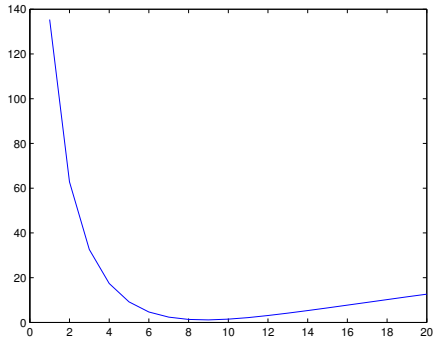
While setting γ to 9 may seem arbitrary, we show here that this effectively produces a near perfect approximation of an exact 2-sparse encoding of the 1-dimensional orientation x over a dictionary of $K = 8$ 1-dimensional atoms:

$$\begin{aligned}
&\mathbf{d}_k = \frac{2k\pi}{8}, 0 \leq k \leq 7, \\
&\boldsymbol{\alpha} \in [0,1]^K, \boldsymbol{\alpha}_k \neq 0 \text{ iff } \mathbf{d}_k \leq x < \mathbf{d}_{k+1}, \\
&x = \mathbf{d}^T \boldsymbol{\alpha}.
\end{aligned}$$

with the convention that $\mathbf{d}_8 = \mathbf{d}_0$. This solves the following constrained optimization



(a) Encoding function with $\gamma = 9$ plotted over 2-sparse continuous encoding of orientation, showing a near-perfect superposition.



(b) Mean squared error between the encoding function and the 2-sparse continuous encoding of orientation, as a function of γ. The minimum among integers is reached for $\gamma = 9$

Figure 2.3: Minimum of the summed squared error is reached for $\gamma = 9$.

problem:

$$\underset{\alpha}{\operatorname{argmin}} \|x - \mathbf{d}^T\boldsymbol{\alpha}\|^2, s.t. \boldsymbol{\alpha} \in [0,1]^K, \|\boldsymbol{\alpha}\|_0 \leq 2$$

The minimum of the summed squared error between $|\cos(\mathbf{x}-\mathbf{d}_k)|_+^\gamma$ (where $|t|_+$ denotes the positive part of t, $\max(t,0)$) and the 2-sparse encoding of angles described above is obtained for $\gamma = 9$, as shown in Fig. (2.3(b)). This widely used implementation of SIFT descriptors is thus in practice a very good approximation of a simple intuitive sparse coding scheme of orientations.

2.2.2 Forming the SIFT descriptors

The coding step described in Sec. (2.2.1) produces an 8-dimensional vector at each location. This output can be viewed as a set of 8 images formed by the activations of

each of the feature components, called *feature maps*. The pooling and subsampling step consists in building histograms of orientations from these feature maps, using cells of 4×4 pixels over the entire image. The histograms can be aligned on the dominant orientation to produce rotation-invariant SIFT descriptors, however this step is often ommitted in image classification pipelines based on the spatial pyramid. A set of 4×4 non-overlapping cells are used to form each SIFT descriptor, by concatenating the histograms for each of the 16 cells and normalizing, yielding a descriptor of dimension $128 = 16 * 8$. Although the selection of *non-overlapping cells* corresponds to subsampling by a factor of 4 within the context of each descriptor, the spatial dimensions of the resulting SIFT descriptors maps depends on how finely spaced the descriptors are; e.g., extracting one SIFT at every pixel yields no subsampling, while spacing the descriptors by 8 pixels (the default setting used in (Lazebnik et al., 2006)) yields a subsampling of 8.

2.3 Adding a third layer

Viewing low- and mid-level vision feature extractors as stackable layers suggests pushing the recursion further and building architectures with three or more layers. This connects vision architectures to *deep learning* models (Bengio, 2007), which rely on a core feature extraction layer (e.g., an RBM (Hinton and Salakhutdinov, 2006) or an autoencoder (Ranzato et al., 2007c; Vincent et al., 2008)) that could potentially be replicated numerous times. However, it is as yet unclear how to determine beforehand the optimal number of layers. In this section, we investigate whether adding more layers would yield any improvements.

Architecture	1	2	1+2
$K_1 = 256, K_2 = 1024$	$\mathbf{82.6 \pm 0.5}$	75.4 ± 0.5	$\mathbf{83.0 \pm 0.6}$
$K_1 = 1024, K_2 = 256$	$\mathbf{84.0 \pm 0.4}$	68.6 ± 0.5	$\mathbf{84.0 \pm 0.5}$
$K_1 = 1024, K_2 = 1024$			$\mathbf{83.9 \pm 0.4}$

Table 2.1: Comparison between 2- and 3-level architectures on the Scenes database. One mid-level layer is composed of a sparse coding module followed by a max pooling module. K_1 and K_2 give the dictionary sizes of each mid-level layer. The second mid-level layer performs much worse than the first one when used by itself, and barely affects classification performance when used jointly with the first one.

The 15-Scenes recognition benchmark (Lazebnik et al., 2006) is used. It is composed of fifteen scene categories, with 200 to 400 images each, and an average image size of 300×250 pixels. Following the usual procedure (Lazebnik et al., 2006; Yang et al., 2009b), we use 100 images per class for training and the rest for testing. The modules composing the baseline architecture are presented in detail in Sec. (3.1); briefly, the architectures compared here comprise low-level layer that extracts dense SIFT descriptors, followed by one or two mid-level feature extraction layers. The mid-level feature extraction layer consists of a sparse coding and a max pooling module. When using a second mid-level feature extraction layer, max pooling is performed over 2×2 neighborhoods of the sparse feature maps. The output of the first, the second, or both mid-level layers are used as input to the classifier, by replicating the max pooling layer to perform pyramidal pooling. Classification is performed using an SVM with an intersection kernel histogram. Hyperparameters are selected by cross-validation using part of the

training data.

Results are shown in Table (2.1). The second mid-level feature extraction layer performs worse than the first mid-level feature extraction layer when used as sole input to the classifier. When used together, the performance is not significantly different than for the first layer alone. Therefore, a single mid-level feature layer is used on top of the low-level feature layer in the remainder of this thesis.

3

COMBINING LAYERS AND MODULES: AN EXTENSIVE EMPIRICAL STUDY OF CLASSIFICATION PERFORMANCE

Since the introduction of bags of features in computer vision (Sivic and Zisserman, 2003), much work has been devoted to improving the baseline performance of a bag-of-words image classification pipeline, usually focusing on tweaking one particular module (e.g., replacing hard vector quantization by soft vector quantization). The goal of this chapter is to determine the relative importance of each module when they are used in combination, and assess to what extent the better performance of a given module is robust to changes in the other modules. We also investigate how best to articulate the low-level feature extraction layer to the mid-level layer, which leads us to propose *macro-features*. Most of the research presented in this chapter has been published in (Boureau et al., 2010a).

3.1 Coding and pooling modules

This section presents some coding and pooling modules proposed in the literature, and discusses their properties.

3.1.1 Notation

Let us first briefly introduce some notation used throughout this thesis. Let I denote an input image. First, low-level descriptors $\boldsymbol{x}_i$ (e.g., SIFT or HOG) are extracted densely at N locations identified with their indices $i = 1, \ldots, N$. Let f and g denote some coding and pooling operators, respectively. M regions of interests are defined on the image (e.g., the $21 = 4 \times 4 + 2 \times 2 + 1$ cells of a three-level spatial pyramid), with $\mathcal{N}_m$ denoting the set of locations/indices within region m. The vector $\boldsymbol{z}$ representing the whole image is obtained by sequentially coding, pooling over all regions, and concatenating:

$$\boldsymbol{\alpha}_i = f(\boldsymbol{x}_i),\ i = 1, \cdots, N \tag{3.1}$$

$$\boldsymbol{h}_m = g\left(\{\boldsymbol{\alpha}_i\}_{i \in \mathcal{N}_m}\right),\ m = 1, \cdots, M \tag{3.2}$$

$$\boldsymbol{z}^T = [\boldsymbol{h}_1^T \cdots \boldsymbol{h}_M^T]. \tag{3.3}$$

The goal is to determine which operators f and g provide the best classification performance using $\boldsymbol{z}$ as input to either a non-linear intersection kernel SVM (Lazebnik et al., 2006), or a linear SVM.

3.1.2 Coding

Coding is performed at each location by applying some operator f chosen to ensure that the resulting codes $\boldsymbol{\alpha}_i$ retain useful information (e.g., input data can be predicted from them), while having some desirable properties (e.g., compactness). Here, we focus on vector quantization and sparse coding, which both minimize some regularized error between inputs and the reconstructions that can be obtained from the codes.

Hard vector quantization (HVQ) is the coding step used in the original bag-of-features framework (Sivic and Zisserman, 2003). f minimizes the distance to a codebook, usually learned by an unsupervised algorithm (e.g., K-means (Lloyd, 1982)). Each x_i is represented by a one-of-K encoding of its cluster assignment:

$$\boldsymbol{\alpha}_i \in \{0,1\}^K, \boldsymbol{\alpha}_{i,j} = 1 \text{ iff } j = \underset{k \leq K}{\operatorname{argmin}} \|\boldsymbol{x}_i - \mathbf{d}_k\|_2^2, \tag{3.4}$$

where $\mathbf{d}_k$ denotes the k-th codeword of the codebook. While hard vector quantization is generally not called sparse coding, it is an extremely sparse code (only *one* component is allowed to be active), where the nonzero coefficient has to be exactly 1:

$$\boldsymbol{\alpha}_i = \underset{\boldsymbol{\alpha} \in \{0,1\}^K}{\operatorname{argmin}} \|\boldsymbol{x} - \mathbf{D}\boldsymbol{\alpha}\|_2^2, \text{ s.t. } \|\boldsymbol{\alpha}\|_0 = 1, \tag{3.5}$$

where ℓ_0 denotes the number of nonzero coefficients (pseudo-norm) of $\boldsymbol{\alpha}$.

Bags of words have been developped in image processing to mimic indexation in a dictionary as closely as possible: identity with a word of the dictionary is replaced with a nearest-neighbor relationship. However, the match between an input patch and the closest codeword is often crude: while texts truly use a finite and discrete corpus of words, image patches present continuous, uncountable variations. Patches are also often arbitrary (e.g., square patches sampled densely on a grid) and do not coincide with meaningful features (e.g., objects or edges) unless preliminary segmentation is performed. Instead, each patch may contain a number of objects, or parts of them. Furthermore, words are rigid, non-scalable discrete units, while image features should allow soft (imperfect) matches, and/or matches to a slightly transformed version of a

codeword, since visual features are invariant to a number of transformations. These are all challenges that soft vector quantization and sparse coding address.

Soft vector quantization uses a softmin to assign scores to codewords that reflect how well they match the input vector:

$$\boldsymbol{\alpha}_{i,j} = \frac{\exp\left(-\beta \|\boldsymbol{x}_i - \mathbf{d}_j\|_2^2\right)}{\sum_{k=1}^{K} \exp\left(-\beta \|\boldsymbol{x}_i - \mathbf{d}_k\|_2^2\right)}, \tag{3.6}$$

where β is a parameter that controls the softness of the soft assignment (hard assignment is the limit when $\beta \to \infty$). This amounts to coding as in the E-step of the expectation-maximization algorithm to learn a Gaussian mixture model, using codewords of the dictionary as centers. This is also similar to to the soft category assignment performed by a multiclass logistic regression classifier, and can be interpreted as a probabilistic version of nearest neighbor search (e.g., see (Goldberger et al., 2004)). Replacing hard matches with soft matches is better suited to cases where an input patch is close to more than one codeword. Soft vector quantization has been shown to improve over hard vector quantization (van Gemert et al., 2010).

However, soft vector quantization still attempts to match each patch to *one* codeword (i.e., distances are computed between the input patch and each of the codewords, one at a time).

Sparse coding (Olshausen and Field, 1997) explains input patches as a linear combination of a small number of codewords. While the resulting code is also continuous, the codewords explain the feature collaboratively, as illustrated in Fig. (3.1), instead of

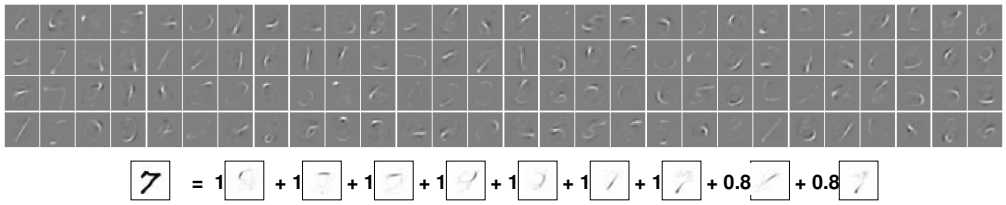


Figure 3.1: Top: filters learned by a sparse energy-based model trained on the MNIST handwritten digit dataset. Bottom: sparse coding performs reconstruction *collaboratively*, so that several localized parts can be combined to reconstruct an input patch. This is in contrast with vector quantization, where each codeword explains the input patch by itself. Figure from (Ranzato et al., 2006)

competitively:

$$\boldsymbol{\alpha}_i = \underset{\boldsymbol{\alpha}\in\mathbb{R}^K}{\operatorname{argmin}} \|\boldsymbol{x}_i - \mathbf{D}\boldsymbol{\alpha}\|_2^2, \text{ s.t. } \|\boldsymbol{\alpha}\|_0 \leq \lambda, \tag{3.7}$$

where $\mathbf{D}$ denotes the dictionary (codebook) of K atoms (codewords) , $\|\boldsymbol{\alpha}\|_0$ denotes the number of nonzero components (ℓ_0 pseudo-norm) of $\boldsymbol{\alpha}$, and integer λ controls the sparsity. What is minimized here is a distance to subspaces instead of a distance to points, and the point corresponding to the final code is generally not one of the codewords, but a point of a subspace generated by a few of these.

The ℓ_0 constraint produces an NP-hard optimization problem and can be relaxed into a tractable ℓ_1 constraint:

$$\boldsymbol{\alpha}_i = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}}\, L_i(\boldsymbol{\alpha}, \mathbf{D}) \triangleq \|\boldsymbol{x}_i - \mathbf{D}\boldsymbol{\alpha}\|_2^2 + \lambda\|\boldsymbol{\alpha}\|_1, \tag{3.8}$$

where $\|\boldsymbol{\alpha}\|_1$ denotes the ℓ_1 norm of $\boldsymbol{\alpha}$, λ is a parameter that controls the sparsity of $\boldsymbol{\alpha}$.

The interpretation in terms of number of nonzero components is lost. The dictionary can be obtained by K-means, or for better performance, trained by minimizing the average of $L_i(\boldsymbol{\alpha}_i, \mathbf{D})$ over all samples, alternatively over $\mathbf{D}$ and the $\boldsymbol{\alpha}_i$. This problem is not jointly convex in $\mathbf{D}$ and $\boldsymbol{\alpha}$ but is convex in each of those parameters when the other is fixed. It is well known that the ℓ_1 penalty induces sparsity and makes the problem tractable (e.g., (Lee et al., 2006; Donoho, 2006; Mairal et al., 2009a)). Sparse coding has been shown to generally perform better than either hard or soft vector quantization for image recognition (Boureau et al., 2010a; Yang et al., 2009b).

3.1.3 Pooling

A pooling operator takes the varying number of codes that are located within M possibly overlapping regions of interest (e.g., the cells of a spatial pyramid), and summarizes them as a single vector of fixed length. The representation for the global image is obtained by concatenating the representations of each region of interest, possibly with a suitable weight. We denote by $\mathcal{N}_m$ the set of locations/indices within region m. In this thesis, we mainly consider the two pooling strategies of average and max pooling.

Average pooling simply computes the average of the codes over the region, and is the pooling method used in the seminal bag-of-features framework (Sivic and Zisserman, 2003):

$$\boldsymbol{h}_m = \frac{1}{|\mathcal{N}_m|} \sum_{i \in \mathcal{N}_m} \boldsymbol{\alpha_i}, \tag{3.9}$$

Note that averaging and using uniform weighting is equivalent (up to a constant multiplicator) to using histograms with weights inversely proportional to the area of the pooling regions, as in (Lazebnik et al., 2006).

Max pooling computes the maximum of each component instead of its average:

$$\boldsymbol{h}_{m,j} = \max_{i \in \mathcal{N}_m} \boldsymbol{\alpha}_{i,j}, \text{ for } j = 1, \ldots, K. \tag{3.10}$$

Yang et al. (Yang et al., 2009b) have obtained state-of-the-art results on several image recognition benchmarks by using sparse coding and max pooling.

3.2 Interaction of coding and pooling modules

This section offers comprehensive comparisons of unsupervised coding schemes, testing all combinations of coding and pooling modules presented in Sec. (3.1). Macrofeatures will be introduced in Sec. (3.3), but some results are included in this section's tables for ease of comparison.

Experiments use the Caltech-101 (Fei-Fei et al., 2004) and Scenes datasets (Lazebnik et al., 2006) as benchmarks. These datasets respectively comprise 101 object categories (plus a "background" category) and fifteen scene categories. Following the usual procedure (Lazebnik et al., 2006; Yang et al., 2009b), we use for each category either 15 training images and 15 testing images, or 30 training images and the rest for testing (with a maximum of 50 test images) on the Caltech-101 dataset, and 100 training images and the rest for testing on the Scenes dataset. Experiments are conducted over 10 random splits of the data, and we report the mean average per-class accuracy and its stan-

dard deviation. Hyperparameters of the model are selected by cross-validation within the training set. The general architecture follows (Lazebnik et al., 2006). Low-level descriptors $\boldsymbol{x}_i$ are 128-dimensional SIFT descriptors (Lowe, 2004) of 16×16 patches. The descriptors are extracted on a dense grid rather than at interest points, since this procedure has been shown to yield superior scene classification (Fei-Fei and Perona, 2005). Pooling regions m comprise the cells of 4×4, 2×2 and 1×1 grids (forming a three-level pyramid). We use the SPAMS toolbox (SPAMS, 2012) to compute sparse codes. Dictionaries for hard and soft vector quantization are obtained with the K-means algorithm, while dictionaries for sparse codes use ℓ_1-regularized reconstruction error during training.

Results are presented in Table (3.1) and Table (3.2). We only show results using 30 training examples per category for the Caltech-101 dataset, since the conclusions are the same when using 15 training examples. The ranking of performance when changing a particular module (e.g., coding) presents a consistent pattern:

- Sparse coding improves over soft quantization, which improves over hard quantization;
- Max pooling almost always improves over average pooling, dramatically so when using a linear SVM;
- The intersection kernel SVM performs similarly or better than the linear SVM.

In particular, the global feature obtained when using hard vector quantization with max pooling achieves high accuracy with a linear classifier, while being *binary*, and merely recording the presence or absence of each codeword in the pools. While much

research has been devoted to devising the best possible coding module, our results show that with linear classification, switching from average to max pooling increases accuracy more than switching from hard quantization to sparse coding. These results could serve as guidelines for the design of future architectures.

For comparison, previously published results obtained using one type of descriptors on the same dataset are shown in Table (3.3). Note that better performance has been reported with multiple descriptor types (e.g., methods using multiple kernel learning have achieved $77.7\% \pm 0.3$ (Gehler and Nowozin, 2009) and $78.0\% \pm 0.3$ (VGG Results URL, 2012; Vedaldi et al., 2009) on Caltech-101 with 30 training examples), or subcategory learning (83% on Caltech-101 (Todorovic and Ahuja, 2008)). The coding and pooling module combinations used in (van Gemert et al., 2010; Yang et al., 2009b) are included in our comparative evaluation (bold numbers in parentheses in Table (3.1), Table (3.2) and Table (3.3)). Overall, our results confirm the experimental findings in these works, except that we do not find superior performance for the linear SVM, compared to the intersection kernel SVM, with sparse codes and max pooling, contrary to Yang et al. (Yang et al., 2009b). Results of our reimplementation are similar to those in (Lazebnik et al., 2006). The better performance than that reported by Van Gemert et al. (van Gemert et al., 2010) or Yang et al. (Yang et al., 2009b) on the Scenes is not surprising since their baseline accuracy for the method in (Lazebnik et al., 2006) is also lower, which they attributed to implementation differences. Discrepancies with results from Yang et al. (Yang et al., 2009b) may arise from their using a differentiable quadratic hinge loss instead of the standard hinge loss in the SVM, and a different type of normalization for SIFT descriptors.

	Average Pool	Max Pool
Results with basic features, SIFT extracted each 8 pixels		
Hard quantization, linear kernel	51.4 ± 0.9 [256]	64.3 ± 0.9 [256]
Hard quantization, intersection kernel	64.2 ± 1.0 [256] **(1)**	64.3 ± 0.9 [256]
Soft quantization, linear kernel	57.9 ± 1.5 [1024]	69.0 ± 0.8 [256]
Soft quantization, intersection kernel	66.1 ± 1.2 [512] **(2)**	70.6 ± 1.0 [1024]
Sparse codes, linear kernel	61.3 ± 1.3 [1024]	$\mathbf{71.5 \pm 1.1}$ [1024] **(3)**
Sparse codes, intersection kernel	70.3 ± 1.3 [1024]	$\mathbf{71.8 \pm 1.0}$ [1024] **(4)**
Results with macrofeatures and denser SIFT sampling		
Hard quantization, linear kernel	55.6 ± 1.6 [256]	70.9 ± 1.0 [1024]
Hard quantization, intersection kernel	68.8 ± 1.4 [512]	70.9 ± 1.0 [1024]
Soft quantization, linear kernel	61.6 ± 1.6 [1024]	71.5 ± 1.0 [1024]
Soft quantization, intersection kernel	70.1 ± 1.3 [1024]	73.2 ± 1.0 [1024]
Sparse codes, linear kernel	65.7 ± 1.4 [1024]	$\mathbf{75.1 \pm 0.9}$ [1024]
Sparse codes, intersection kernel	73.7 ± 1.3 [1024]	$\mathbf{75.7 \pm 1.1}$ [1024]

Table 3.1: Average recognition rate on the Caltech-101 benchmark, using 30 training examples, for various combinations of coding, pooling, and classifier types. The codebook size shown inside brackets is the one that gives the best results among 256, 512 and 1024. Linear and histogram intersection kernels are identical when using hard quantization with max pooling (since taking the minimum or the product is the same for binary vectors), but results have been included for both to preserve the symmetry of the table. Top: Results with the baseline SIFT sampling density of 8 pixels and standard features. Bottom: Results with the set of parameters for SIFT sampling density and macrofeatures giving the best performance for sparse coding.

	Average Pool	Max Pool
Results with basic features, SIFT extracted each 8 pixels		
Hard quantization, linear kernel	73.9 ± 0.9 [1024]	80.1 ± 0.6 [1024]
Hard quantization, intersection kernel	80.8 ± 0.4 [256] **(1)**	80.1 ± 0.6 [1024]
Soft quantization, linear kernel	75.6 ± 0.5 [1024]	81.4 ± 0.6 [1024]
Soft quantization, intersection kernel	81.2 ± 0.4 [1024] **(2)**	83.0 ± 0.7 [1024]
Sparse codes, linear kernel	76.9 ± 0.6 [1024]	83.1 ± 0.6 [1024] **(3)**
Sparse codes, intersection kernel	83.2 ± 0.4 [1024]	$\mathbf{84.1 \pm 0.5}$ [1024] **(4)**
Results with macrofeatures and denser SIFT sampling		
Hard quantization, linear kernel	74.0 ± 0.5 [1024]	80.1 ± 0.5 [1024]
Hard quantization, intersection kernel	81.0 ± 0.5 [1024]	80.1 ± 0.5 [1024]
Soft quantization, linear kernel	76.4 ± 0.7 [1024]	81.5 ± 0.4 [1024]
Soft quantization, intersection kernel	81.8 ± 0.4 [1024]	83.0 ± 0.4 [1024]
Sparse codes, linear kernel	78.2 ± 0.7 [1024]	83.6 ± 0.4 [1024]
Sparse codes, intersection kernel	83.5 ± 0.4 [1024]	$\mathbf{84.3 \pm 0.5}$ [1024]

Table 3.2: Average recognition rate on the 15-Scenes benchmarks, using 100 training examples, for various combinations of coding, pooling, and classifier types. The codebook size shown inside brackets is the one that gives the best results among 256, 512 and 1024. Linear and histogram intersection kernels are identical when using hard quantization with max pooling (since taking the minimum or the product is the same for binary vectors), but results have been included for both to preserve the symmetry of the table. Top: Results with the baseline SIFT sampling density of 8 pixels and standard features. Bottom: Results with the set of parameters for SIFT sampling density and macrofeatures giving the best performance for sparse coding.

Method	C101, 15 tr.	C101, 30 tr.	Scenes
Nearest neighbor + spatial correspondence (Boiman et al., 2008)	65.0 ± 1.1	70.4	–
Similarity-preserving sparse coding (Gao et al., 2010)	–	–	89.8 ± 0.5
Fast image search for learned metrics (Jain et al., 2008)	61.0	69.6	–
(1) SP + hard quantization + kernel SVM (Lazebnik et al., 2006)	56.4	64.4 ± 0.8	81.4 ± 0.5
(2) SP + soft quantization + kernel SVM (van Gemert et al., 2010)	–	64.1 ± 1.2	76.7 ± 0.4
SP + locality-constrained linear codes (Wang et al., 2010)	–	73.4	–
(3) SP + sparse codes + max pooling + linear SVM (Yang et al., 2009b)	$\mathbf{67.0 \pm 0.5}$	$\mathbf{73.2 \pm 0.5}$	80.3 ± 0.9
(4) SP + sparse codes + max pooling + kernel SVM (Yang et al., 2009b)	60.4±1.0	–	77.7±0.7
kNN-SVM (Zhang et al., 2006)	59.1 ± 0.6	66.2 ± 0.5	-
SP + Gaussian mixture (Zhou et al., 2008)	–	–	$\mathbf{84.1 \pm 0.5}$

Table 3.3: Performance of several schemes using a single type of descriptors. Italics indicate results published after our CVPR paper (Boureau et al., 2010a). Bold numbers in parentheses preceding the method description indicate methods reimplemented here. 15tr., 30tr.: 15 and 30 training images per category, respectively. SP: spatial pyramid.

3.3 Macrofeatures

In convolutional neural networks (e.g., (Lee et al., 2009; Ranzato et al., 2007b)), spatial neighborhoods of low-level features are encoded jointly. On the other hand, codewords in bag-of-features methods usually encode low-level features at a single location (see Fig. (3.2)). We propose to adapt the joint encoding scheme to the spatial pyramid framework.

Jointly encoding L descriptors in a local spatial neighborhood $\mathcal{L}_i$ amounts to replacing Eq. (3.1)) by:

$$\boldsymbol{\alpha}_i = f([\boldsymbol{x}_{i_1}^T \cdots \boldsymbol{x}_{i_L}^T]^T), \;\; i_1, \cdots, i_L \in \mathcal{L}_i. \tag{3.11}$$

We call *macrofeatures* vectors that jointly encode a small neighborhood of SIFT descriptors. The encoded neighborhoods are squares determined by two parameters: the number of neighboring SIFT descriptors considered along each spatial dimension (e.g., 2×2 square in Fig. (3.2)), and a macrofeature subsampling stride which gives the number of pixels to skip between neighboring SIFT descriptors within the macrofeature. This is distinct from the grid subsampling stride that controls how finely SIFT descriptors are extracted over the input image. For example, a 3×3 macrofeature with a macrofeature subsampling stride of 6 pixels, and a grid subsampling stride of 3 pixels, jointly encodes 9 descriptors, skipping every other SIFT descriptor along columns and rows over a neighborhood of 6×6 descriptors.

We have experimented with different macrofeature parameters, and denser sampling of the underlying SIFT descriptor map (e.g., extracting SIFT every 4 pixels instead of 8 pixels as in the baseline of (Lazebnik et al., 2006)). We have tested grid subsam-

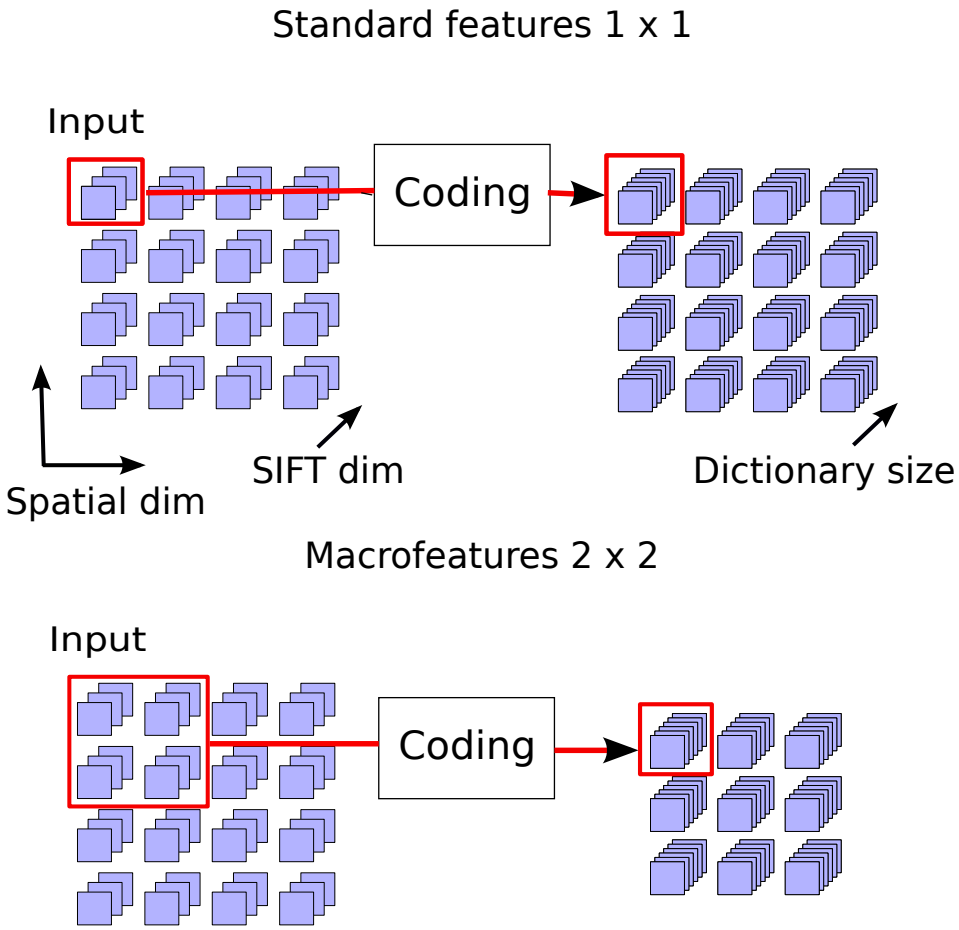


Figure 3.2: Standard features encode the SIFT features at a single spatial point. Macrofeatures jointly encode small spatial neighborhoods of SIFT features (i.e., the input of the coding module is formed by concatenating nearby SIFT descriptors).

pling strides ranging from 2 to 10, macrofeatures of side length 2 to 4 and subsampling strides 1 to 4 times the base grid subsampling stride. When using sparse coding and max pooling, the best parameters (selected by cross-validation within the training set) for SIFT grid subsampling stride, and macrofeature side length and subsampling stride are respectively of 4 pixels, 2 descriptors, and 16 pixels for the Caltech-101 dataset, and 8 pixels, 2 descriptors, 8 pixels for the Scenes dataset. Our results (Table (3.1) and Table (3.2), bottom) show that large improvements can be gained on the Caltech-101 benchmark, by merely sampling SIFT descriptors more finely, and jointly representing nearby descriptors, yielding a classification accuracy of 75.7%, which to the best of our knowledge outperformed all classification schemes using a single type of low-level descriptor at the time of publication of our CVPR paper (Boureau et al., 2010a). However, we have not found finer sampling and joint encoding to help recognition significantly on the Scenes dataset. More comprehensive results for Caltech-101 when using sparse coding, max pooling and a linear classifier are presented in Table (3.4) to show the separate influence of each of these hyperparameters.

On the Scenes dataset, sampling features on a finer grid slightly damages performance, with both max pooling, while using macrofeatures of size 2×2 seems to slightly improve results (see Table (3.5) and Fig. (3.3)), but the differences in performance are not significant.

3.4 Choosing the dictionary

Experiments in this section look at the influence of the dictionary, answering two questions: (1) are the relative performances of coding and pooling methods always the same

Standard Feature1 × 11					
Grid subs.	2	3	4	6	8
RF	16	16	16	16	16
	74.4 ± 0.9	**74.5±1.1**	73.8 ± 1.0	73.3 ± 1.0	71.5 ± 1.1
Grid subsampling 8					
# descriptors	1×1	2×2	3×3	4×4	5×5
RF	16	24	32	40	48
	71.5 ± 1.1	72.6 ± 1.0	**73.3±1.1**	73.2 ± 1.0	73.3 ± 1.0
Grid subsampling 4, 2×2 adjacent descriptors					
Stride	4	8	12	16	20
RF	20	24	28	32	36
	73.8 ± 0.8	74.4 ± 0.9	74.6 ± 1.0	**75.1±0.9**	74.7 ± 1.1
Grid subsampling 3					
# descriptors	2×2	3×3	4×4	2×2	3×3
Stride	3	3	3	12	12
RF	19	22	25	28	40
	73.6 ± 0.9	73.7 ± 0.7	74.0 ± 1.0	**74.8±0.9**	74.5 ± 1.2

Table 3.4: Mean accuracy on the Caltech 101 dataset, using 1024 codewords, max pooling, linear classifier, and 30 training examples per category. # descriptors: number of SIFT descriptors jointly encoded into one macrofeature. Grid subsampling: number of pixels separating one macrofeature from the next. Stride (macrofeature subsampling stride): number of pixels separating two SIFT descriptors used as input of a macrofeature. RF (receptive field): side of the square spanned by a macrofeature, in pixels; function of the other parameters.

Grid resolution	4	6	8
Linear	81.8 ± 0.6	82.0 ± 0.6	$\mathbf{83.1 \pm 0.6}$
Intersect	82.4 ± 0.5	82.7 ± 0.6	$\mathbf{84.1 \pm 0.5}$

Table 3.5: Varying grid resolution on the Scenes dataset, with linear or intersection kernels. Codebook size 1024.

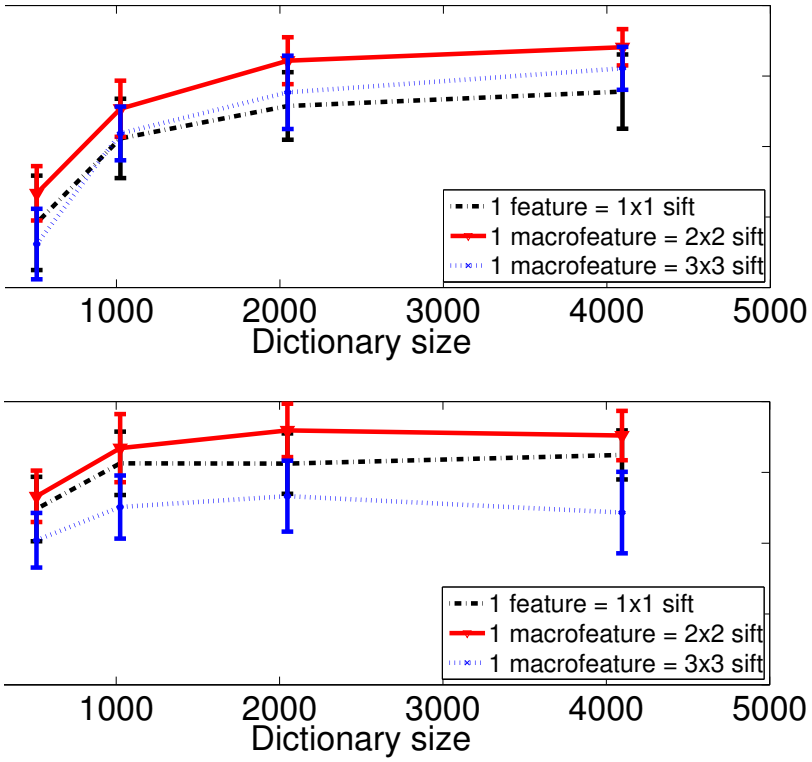


Figure 3.3: Representing small 2×2 neighborhoods of SIFT jointly by one sparse code leads to slightly better results on the Scenes dataset than 1×1 or 3×3 neighborhoods, with both linear (top) and intersection (bottom) kernels.

regardless of dictionary size, and (2) does the superiority of sparse coding vanish if the dictionary is not trained differently than for vector quantization?

3.4.1 Dictionary size

The codebook size does not have the same influence according to the type of feature used (quantization or sparse coding). Fig. (3.4) plots recognition accuracy on the Caltech 101 dataset for several dictionary sizes, using average pooling. Hard and soft quantization perform best for a fairly small dictionary size, while the performance stays stable or increases slightly with sparse coding combined with average pooling.

Fig. (3.5) and Fig. (3.6) compare sparse coding combined with several types of pooling and kernels, on the Scenes and Caltech 101 datasets, respectively. Recognition accuracy increases more sharply with max pooling. On the Scenes dataset (Fig. (3.5)), a larger range of dictionary sizes has been tested, showing that recognition accuracy drops with average pooling when the dictionary gets too large ($K > 1024$).

3.4.2 How important is dictionary training?

It could be argued that sparse coding performs better than vector quantization because the dictionary obtained with ℓ_1-regularized reconstruction error is better than the one obtained with K-means. We have run experiments to compare hard and soft vector quantization, and sparse coding, over the same K-means dictionary, as well as sparse coding over a dictionary trained with a sparse coding penalty. For these experiments, a grid has sometimes been used for pooling instead of a pyramid. Average pooling and an intersection kernel are used.

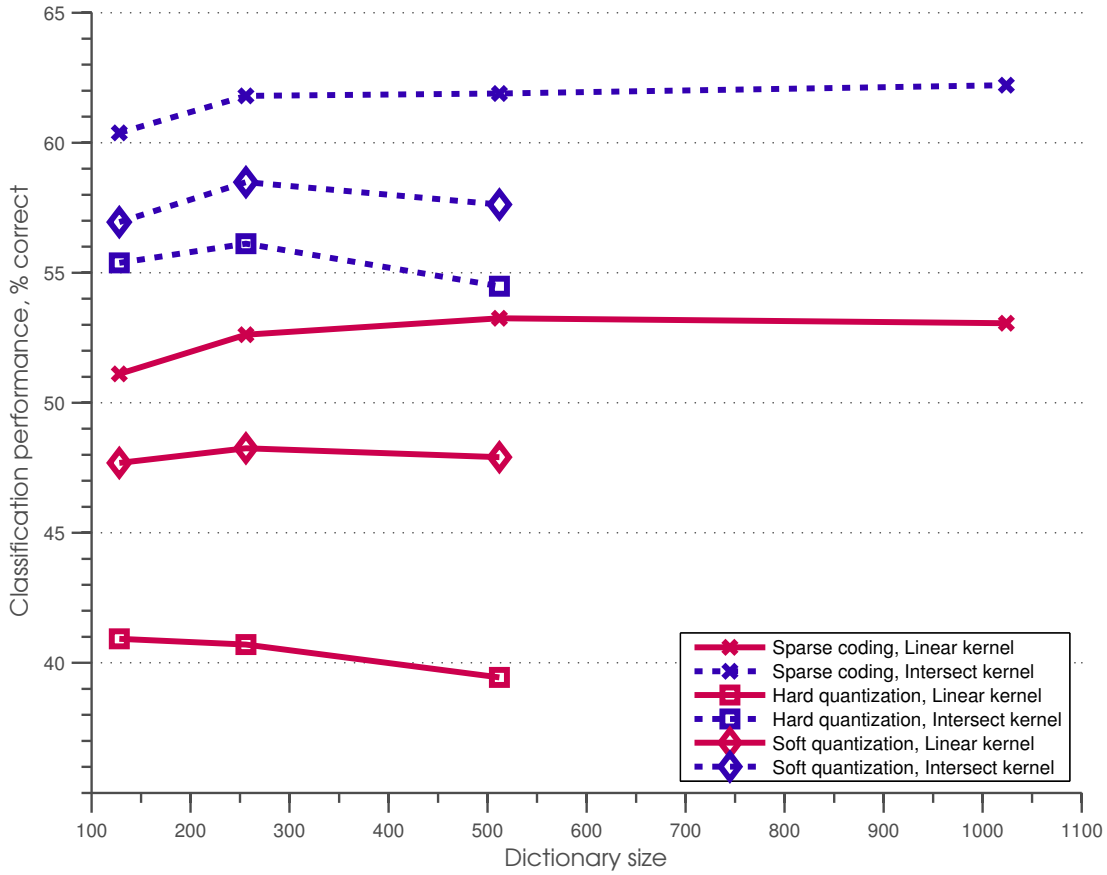


Figure 3.4: Recognition accuracy on the Caltech 101 database with 15 training examples with average pooling on a 4×4 grid. With vector quantization, the best performance is obtained with a small dictionary. Performance stays stable with sparse codes when increasing dictionary size. For all classifiers and dictionary sizes, sparse coding performs best and hard quantization performs worst, with soft quantization in between. The intersection kernel is clearly better than the linear kernel when average pooling is used. The worst performance with an intersection kernel classifier (three top curves, dotted) is better than the best performance with a linear classifier (three bottom curves, solid).

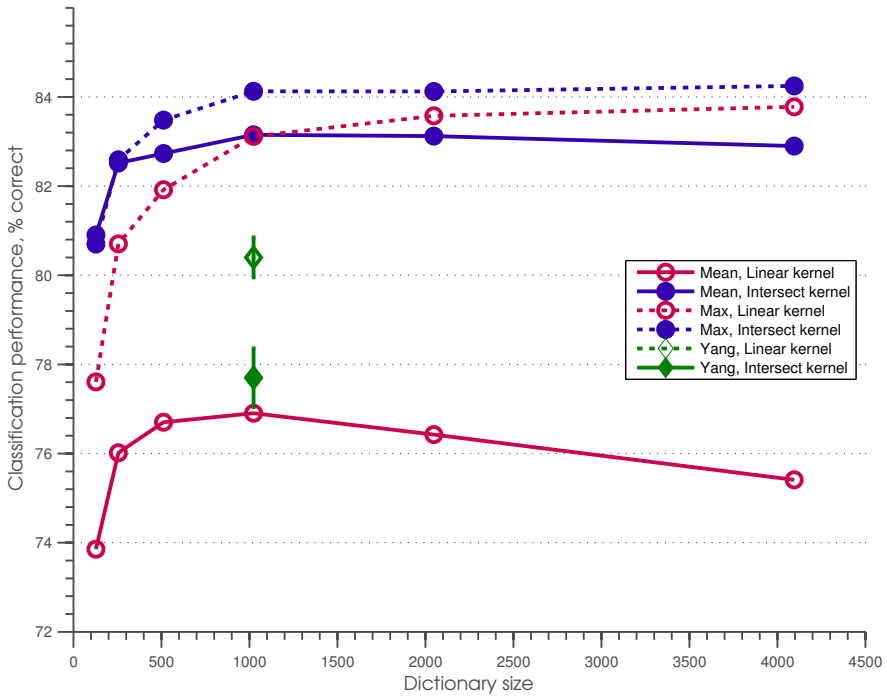


Figure 3.5: Performance on the Scenes dataset using sparse codes. Each curve plots performance against dictionary size for a specific combination of pooling (taking the average or the max over the neighborhood) and classifier (SVM with linear or intersection kernel). 1) Performance of the max pooling (dotted lines) is consistently higher than average pooling (solid lines), but the gap is much less significant with intersection kernel (closed symbols) than with linear kernel (open symbols). Slope is steeper with the max/linear combination than if either the pooling or the kernel type is changed. 2) Intersection kernel (closed symbols) performs generally better than linear kernels (open symbols), especially with average pooling (solid lines) or with small dictionary sizes. This is contrary to Yang's results (Yang et al., 2009b) where intersection kernels (bottom, closed diamond) perform noticeably worse than linear kernels (top, open diamond).

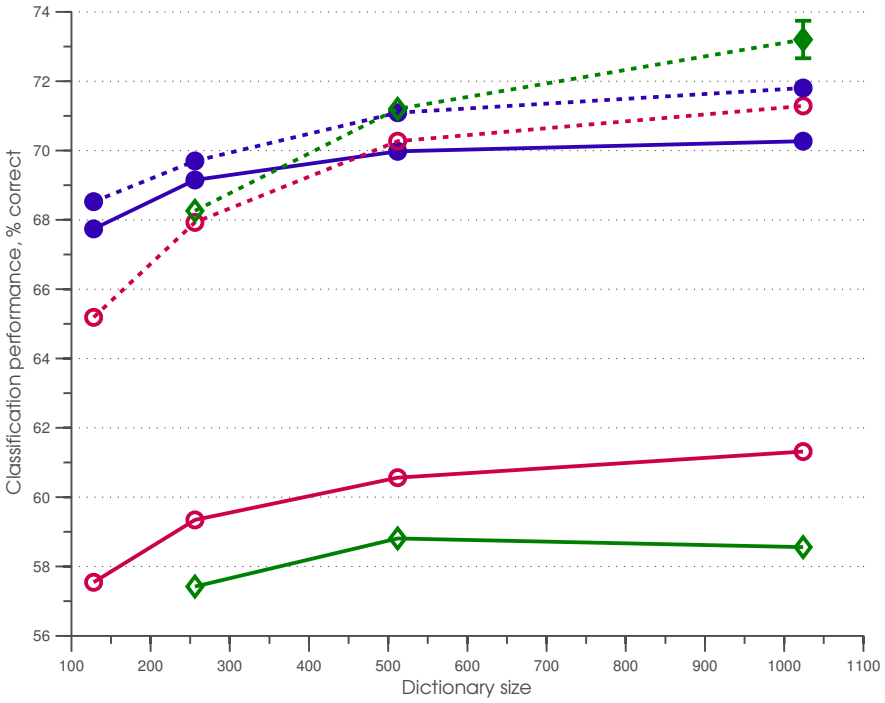


Figure 3.6: Performance on the Caltech 101 dataset using sparse codes and 30 training images per class. Each curve plots performance against dictionary size for a specific combination of pooling (taking the average or the max over the neighborhood) and classifier (SVM with linear or intersection kernel). 1) Performance of the max pooling (dotted lines) is consistently higher than average pooling (solid lines), but the gap is much less significant with intersection kernel (closed symbols) than with linear kernel (open symbols). Slope is steeper with the max/linear combination than if either the pooling or the kernel type is changed. 2) Intersection kernel (closed symbols) performs generally better than linear kernels (open symbols), especially with average pooling (solid lines) or with small dictionary sizes. This is contrary to Yang et al's results (Yang et al., 2009b) where intersection kernels (not shown in this plot) perform noticeably worse than linear kernels.

Results shown in Table (3.6) and Table (3.7) show that sparse coding performs better than hard or soft vector quantization when the same dictionary is used for all coding types. Compared to hard vector quantization, *most* of the improvement is due to sparse coding itself, with the better dictionary improving performance less substantially.

Coding	15 tr.	30 tr.
Hard vector quantization	55.9	63.8
Soft vector quantization	57.9	65.6
Sparse coding		
on k-mean centers	60.7	66.2
on learned dictionary	61.7	68.1

Table 3.6: Recognition accuracy on the Caltech-101 dataset, 4x4 grid, dictionary of size $K = 200$, average pooling, intersection kernel, using 15 or 30 training images per class.

More extensive experiments conducted by Coates and Ng (Coates and Ng, 2011) on other benchmarks compare dictionaries composed of atoms with random values, randomly selected training patches, or optimized to minimize the same loss as during coding. The conclusion is also that optimizing the dictionary does not contribute as much to performance as choosing a better coding algorithm.

3.5 Conclusion

By deconstructing the mid-level coding step of a well-accepted recognition architecture, it appears that any parameter in the architecture can contribute to recognition per-

Coding	4x4 grid	4x4 pyramid
Hard vector quantization	79.1	80.9 ± 0.2
Soft vector quantization	79.8	81.5 ± 0.5
Sparse coding		
on k-mean centers	80.4	81.9 ± 0.6
on learned dictionary	81.0	82.3 ± 0.4

Table 3.7: Recognition accuracy on the Scenes dataset, dictionary of size $K = 200$, using either a 4×4 grid or a 4×4 pyramid, average pooling, intersection kernel.

formance; in particular, surprisingly large performance increases can be obtained by merely sampling the low-level descriptor map more finely, and representing neighboring descriptors jointly. Conversely, the choice of the dictionary does not appear as critical as the choice of the coding and pooling operators. Some of our findings are robust to many changes in surrounding modules: sparse coding outperforms soft vector quantization, which outperforms hard vector quantization; max pooling always performs better than average pooling in our settings.

4

COMPARING MAX AND AVERAGE POOLING

One of the most striking results of our comparative evaluation in the previous chapter is that the superiority of max pooling over average pooling generalizes to many combinations of coding schemes and classifiers. Several authors have already stressed the efficiency of max pooling (Jarrett et al., 2009; Yang et al., 2009b), but they have not given theoretical explanations to their findings. In this chapter, we study max and average pooling in more detail theoretically and experimentally. The work presented in this chapter has been published in (Boureau et al., 2010a) and (Boureau et al., 2010b).

4.1 Introduction

In general terms, the objective of pooling is to transform the joint feature representation into a new, more usable one that preserves important information while discarding irrelevant detail, the crux of the matter being to determine what falls in which category. For example, the assumption underlying the computation of a histogram is that the average feature activation matters, but exact spatial localization does not. Achieving invariance to changes in position or lighting conditions, robustness to clutter, and compactness of representation, are all common goals of pooling.

The success of the spatial pyramid model (Lazebnik et al., 2006), which obtains

large increases in performance by performing pooling over the cells of a spatial pyramid rather than over the whole image as in plain bag-of-features models (Zhang et al., 2007), illustrates the importance of the spatial structure of pooling neighborhoods. Perhaps more intriguing is the dramatic influence of the way pooling is performed once a given region of interest has been chosen. Thus, Jarrett et al. (Jarrett et al., 2009) have shown that pooling type matters more than careful unsupervised pretraining of features for classification problems with little training data, obtaining good results with random features when appropriate pooling is used. Yang et al. (Yang et al., 2009b) report much better classification performance on several object or scene classification benchmarks when using the maximum value of a feature rather than its average to summarize its activity over a region of interest. But no theoretical justification of these findings is given. In the previous chapter, we have shown that using max pooling on hard-vector quantized features (which produces a binary vector that records the presence of a feature in the pool) in a spatial pyramid brings the performance of linear classification to the level of that obtained by Lazebnik et al. (Lazebnik et al., 2006) with an intersection kernel, even though the resulting feature is binary. However, it remains unclear why max pooling performs well in a large variety of settings, and indeed whether similar or different factors come into play in each case.

This chapter attempts to fill the gap and conducts a thorough theoretical investigation of pooling. We compare different pooling operations in a categorization context, and examine how the behavior of the corresponding statistics may translate into easier or harder subsequent classification. We provide experiments in the context of visual object recognition, but the analysis applies to all tasks which incorporate some form

of pooling (e.g., text processing from which the bag-of-features method was originally adapted). The main contributions of this chapter are (1) an extensive analytical study of the discriminative powers of different pooling operations, (2) the discrimination of several factors affecting pooling performance, including smoothing and sparsity of the features, (3) the unification of several popular pooling types as belonging to a single continuum.

4.2 Pooling as extracting a statistic

The pyramid match kernel (Grauman and Darrell, 2005) and the spatial pyramid (Lazebnik et al., 2006) have been formulated as better ways to look for correspondances between images, by allowing matches at varying degrees of granularity in feature space (pyramid match kernel), or by taking some coarse spatial information into account to allow two features to match (spatial pyramid). In that view, pooling is relaxing matching constraints to make correspondances more robust to small deformations. Looking for correspondances is still a fertile area of research, and recent work following this direction has obtained state-of-the-art results in object recognition, at some computational cost (Duchenne et al., 2011).

But pooling also extracts a statistic over a given sample. If the goal is to discriminate between two classes, the class-conditional statistics extracted should be different. For example, if the local features are distributed according to Gaussians of different means for each class, then the average should be discriminative. On the other hand, if two Gaussian samples only differ through their variance, then the average is not discriminative, but max is: for a Gaussian distribution, a classical result is that the expectation of

the max over P samples from a distribution of variance σ^2 grows asymptotically (when $P \to \infty$) like $\sqrt{2\sigma^2 \log(P)}$. Thus, the separation of the maxima over two Gaussian samples increases indefinitely with sample size if their standard deviations are different (albeit at a slow rate). Another way to discriminate between Gaussian distributions is to take the average of the absolute value, the positive part, or the square (or any even power) of the local features; e.g., for a Gaussian of mean μ, $\mathbb{E}(|\mathbf{x} - \mu|) = \sigma\sqrt{2/\pi}$. Thus, the average becomes discriminative if a suitable non-linearity is applied first. This type of considerations may explain the crucial importance of taking the absolute value or positive part of the features before average pooling, in the experiments of Jarrett et al. (Jarrett et al., 2009), even though the feature distributions cannot be assumed to be Gaussian.

The statistic may also have dependencies on the sample size, which is usually influenced by the spatial size of the pooling neighborhood (e.g., the smaller cells of a spatial pyramid compared to the whole image). It is intuitively clear that the maximum of a sample often increases with the sample size, while the average does not change (the estimate of the average usually has larger variance with a small sample size, but the expected average is the same). Thus, there may be a purely statistical component to the improvement seen with max pooling when using pyramids instead of plain bags of features. Max pooling differs from average pooling in two important ways:

- the maximum over a pool of smaller cardinality is not merely an estimator of the maximum over a larger pool;
- the variance of the maximum is not generally inversely proportional to pool cardinality, so that summing over several estimates (one for each smaller pool) can

	Caltech 101		15 Scenes	
Pyramid	1×1	2×2	1×1	2×2
Avg, random	31.7 ± 1.0	29.5 ± 0.5	71.0 ± 0.8	69.4 ± 0.8
Avg, spatial		43.2 ± 1.4		73.2 ± 0.7
Max, random	26.2 ± 0.7	33.1 ± 0.9	69.5 ± 0.6	72.8 ± 0.3
Max, spatial		50.7 ± 0.8		77.2 ± 0.6

Table 4.1: Classification accuracy for different sets of pools and pooling operators. Spatial: the pools are cells of a spatial pyramid. Random: features have been randomly scrambled before pooling, effectively removing all spatial information.

provide a smoother output than if pooling had merely been performed over the merged smaller pools.

The next sections are devoted to more detailed modeling, but a simple experiment can demonstrate this effect. We compare three types of pooling procedures: standard whole-image and two-level pyramid pooling, and random two-level pyramid pooling, where local features are randomly permuted before being pooled, with a new random permutation being picked for each image: all spatial information is removed, but the pools have a smaller number of samples in the finer cells of the pyramid.

This experiment shares most settings with those in the previous chapter, using SIFT features extracted densely every 8 pixels, and encoded by hard quantization over a codebook of size 256 for Caltech-101, 1024 for the Scenes. The pooled features are concatenated and classified with a linear SVM, trained on 30 and 100 examples per category for Caltech-101 and the Scenes, respectively.

Results are shown in Table (4.1). Predictably, keeping spatial information improves performance in all cases. But with max pooling, a substantial part of the increase in accuracy seen when using a two-level pyramid instead of a plain bag of features is still present when locations are randomly shuffled. Conversely, the performance of average pooling tends to deteriorate with the pyramid, since the added smaller, random pools only contribute noisier, redundant information.

4.3 Modeling pooling

Consider a two-class categorization problem. Intuitively, classification is easier if the distributions from which points of the two classes are drawn have no overlap. In fact, if the distributions are simply shifted versions of one another (e.g., two Gaussian distributions with same variance), linear separability increases monotonically with the magnitude of the shift (e.g., with the distance between the means of two Gaussian distributions of same variance) (Bruckstein and Cover, 1985). In this section, we examine how the choice of the pooling operator affects the separability of the resulting distributions. Let us start with a caveat. Several distributions are at play here: the distributions we are trying to separate for classification are distributions of multi-dimensional feature vectors representing one image, obtained after pooling and concatenating — the components of each of these feature vectors are obtained by estimating parameters on 'lower-level' distributions, e.g., distributions of local feature vectors within a pooling cell of a given image, as discussed in the previous section.

We separately look at binary and continuous codes.

4.3.1 Pooling binary features

This section deals with pooling over binary codes (e.g., one-of-K codes obtained by vector quantization in bag-of-features models).

Model

Let us examine the contribution of a single feature in a bag-of-features representation — i.e., if the unpooled data is a $P \times K$ matrix of one-of-K codes taken at P locations, we extract a single P-dimensional column $\boldsymbol{v}$ of 0s and 1s, indicating the absence or presence of the feature at each location. Using the notation from Sec. (3.1.1), the $\boldsymbol{\alpha}_i$ would be K-dimensional rows from that matrix, with a single 1 per row.

For simplicity, we model the P components of $\boldsymbol{v}$ as i.i.d. Bernoulli random variables. The independence assumption is clearly false since nearby image features are strongly correlated, but the analysis of this simple model nonetheless yields useful predictions that can be verified empirically. The vector $\boldsymbol{v}$ is reduced by a pooling operator g to a single scalar $g(\boldsymbol{v})$ — which would be one component of the K-dimensional representation using all features, e.g., one bin in a histogram. With the notation from Sec. (3.1.1): $g(\boldsymbol{v}) = \boldsymbol{h}_{m,j} = g\left(\{\boldsymbol{\alpha}_{i,j}\}_{i \in \mathcal{N}_m}\right)$ and $P = |\mathcal{N}_m|$, if the contribution of feature j in pool m is being examined. We consider two pooling operators: average pooling $g_a(\boldsymbol{v}) = \frac{1}{P}\sum_{i=1}^{P} \boldsymbol{v}_i$, and max pooling $g_m(\boldsymbol{v}) = \max_i \boldsymbol{v}_i$.

Distribution separability

Given two classes C_1 and C_2, we examine the separation of conditional distributions $p(g_m|C_1)$ and $p(g_m|C_2)$, and $p(g_a|C_1)$ and $p(g_a|C_2)$. While separability based jointly on k features does not require separability for each individual feature, increased separability

of each marginal feature distribution generally leads to better joint separability. Viewing separability as a signal-to-noise problem, better separability can be achieved by either increasing the distance between the means of the two class-conditional distributions, or reducing their standard deviation.

We first consider average pooling. The sum over P i.i.d. Bernoulli variables of mean ξ follows a binomial distribution $B(P, \xi)$. Consequently, the distribution of g_a is a scaled-down version of the binomial distribution, with mean $\mu_a = \xi$, and variance $\sigma_a^2 = \xi(1-\xi)/P$. The expected value of g_a is independent of sample size P, and the variance decreases like $1/P$; therefore the separation ratio of means' difference over standard deviation decreases monotonically like $1/\sqrt{P}$. Thus, it is always better to take into account all available samples of a given spatial pool in the computation of the average.

Max pooling is slightly less straightforward, so we examine means' separation and variance separately in the next two sections.

Means' separation of max-pooled features

g_m is a Bernoulli variable of mean $\mu_m = 1 - (1-\xi)^P$ and variance $\sigma_m^2 = (1 - (1 - \xi)^P)(1-\xi)^P$. The mean increases monotonically from 0 to 1 with sample size P. Let ϕ denote the separation of class-conditional expectations of max-pooled features,

$$\phi(P) \triangleq |\mathbb{E}(g_m|C_1) - \mathbb{E}(g_m|C_2)| = |(1-\xi_2)^P - (1-\xi_1)^P|, \tag{4.1}$$

where $\xi_1 \triangleq \mathbb{P}(\boldsymbol{v}_i = 1|C_1)$ and $\xi_2 \triangleq \mathbb{P}(\boldsymbol{v}_i = 1|C_2)$. We abuse notation by using ϕ to refer both to the function defined on sample cardinality P and its extension to $\mathbb{R}$. It is

easy to show that ϕ is increasing between 0 and

$$P_M \triangleq \left| \log \left(\frac{\log(1-\xi_2)}{\log(1-\xi_1)} \right) / \log \left(\frac{1-\xi_1}{1-\xi_2} \right) \right|, \tag{4.2}$$

and decreasing between P_M and ∞, with $\lim_0 \phi = \lim_\infty \phi = 0$.

Noting that $\phi(1) = |\xi_1 - \xi_2|$ is the distance between the class-conditional expectations of average-pooled features, there exists a range of pooling cardinalities for which the distance is greater with max pooling than average pooling if and only if $P_M > 1$. Assuming $\xi_1 > \xi_2$, without loss of generality, it is easy to show[1] that:

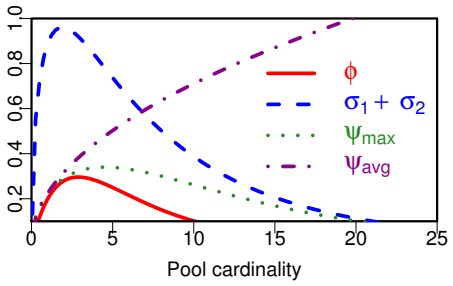
$$P_M \leq 1 \Rightarrow \xi_1 > 1 - 1/e > 0.63. \tag{4.3}$$

$\xi_1 > 0.63$ means that the feature is selected to represent more than half the patches on average, which in practice does not happen in usual bag-of-features contexts, where codebooks comprise more than a hundred codewords.

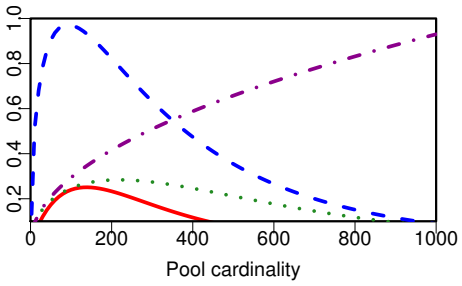
Variance of max-pooled features

The variance of the max-pooled feature is $\sigma_m^2 = (1-(1-\xi)^P)(1-\xi)^P$. A simple analysis of the continuous extension of this function to real numbers shows that it has limit 0 at 0 and ∞, and is increasing then decreasing, reaching its maximum of 0.5 at $\log(2)/|\log(1-\xi)|$. The increase of the variance can play against the better separation of the expectations of the max-pooled feature activation, when parameter values ξ_1 and ξ_2 are too close for the two classes. Several regimes for the variation of means separation and standard deviations are shown in Fig. (4.1).

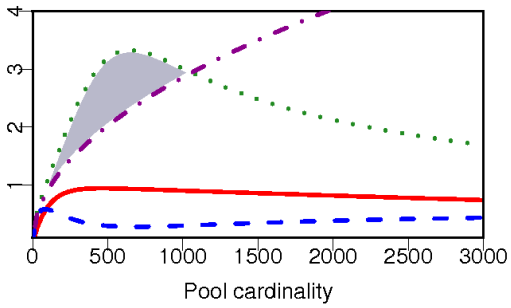
[1] PROOF: Let $\chi(x) \triangleq x \log(x), x \in \mathbb{R}$. $P_M \leq 1 \Leftrightarrow \chi(1-\xi_1) > \chi(1-\xi_2)$. Since $1-\xi_1 < 1-\xi_2$, and χ is decreasing on $[0, 1/e]$ and increasing on $[1/e, 1]$, it follows that $P_M \leq 1 \Rightarrow \xi_1 > 1 - 1/e$. □



(a) $\xi_1 = 0.4, \xi_2 = 0.2$



(b) $\xi_1 = 1.10^{-2}, \xi_2 = 5.10^{-3}$



(c) $\xi_1 = 1.10^{-2}, \xi_2 = 1.10^{-4}$

Figure 4.1: $\phi(P) = |(1-\xi_1)^P - (1-\xi_2)^P|$, σ_1 and σ_2 denote the distance between the expectations of the max-pooled features of mean activation ξ_1 and ξ_2, and their standard deviations, respectively. $\psi_{max} = \phi/(\sigma_1 + \sigma_2)$ and $\psi_{avg} = |\xi_1 - \xi_2|.\sqrt{P}/(\sqrt{\xi_1.(1-\xi_1)} + \sqrt{\xi_2.(1-\xi_2)})$ give a measure of separability for max and average pooling. ϕ reaches its peak at smaller cardinalities than ψ_{max}. (a) When features have relatively large activations, the peak of separability is obtained for small cardinalities (b) With sparser feature activations, the range of the peak is much larger (note the change of scale in the x axis). (c) When one feature is much sparser than the other, ψ_{max} can be larger than ψ_{avg} for some cardinalities (shaded area). Best viewed in color.

Conclusions and predictions

Our simplified analysis leads to several predictions:

- Max pooling is particularly well suited to the separation of features that are very sparse (i.e., have a very low probability of being active)
- Using all available samples to perform the pooling may not be optimal
- The optimal pooling cardinality should increase with dictionary size

The first point can be formalized by observing that the characteristic pooling cardinality $|1/\log(1-\xi)|$ ($\approx 1/\xi$ in the case $\xi \ll 1$), scales the transition to the asymptotic regime (low variance, high probability of activation): the maximum of the variance is reached at $P = \log(2)/|\log(1-\xi)|$, and:

$$\mathbb{P}(g_m(\boldsymbol{v}) = 1) > \lambda \Leftrightarrow P > \frac{\log(1-\lambda)}{\log(1-\xi)}. \tag{4.4}$$

Consequently, the range of cardinalities for which max pooling achieves good separation between two classes doubles if the probability of activation of the feature for both classes is divided by two. A particularly favorable regime is $\xi_2 \ll \xi_1 \ll 1$ — that is, a feature which is rare, but relatively much more frequent in one of the two classes; in that case, both classes reach their asymptotic regime for very different sample cardinalities ($1/\xi_1$ and $1/\xi_2$).

The increase of optimal pooling cardinality with dictionary size is related to the link underlined above between the sparsity of the features (defined here as the probability of them being 0) and the discriminative power of max-pooling, since the expected feature

activations sum to one in the general bag-of-features setting (exactly one feature is activated at each location), resulting in a mean expected activation of $1/K$ with a K-word codebook. Thus, K gives an order of magnitude for the characteristic cardinality scale of the transition to the asymptotic regime, for a large enough codebook.

The experiments in Sec. (4.2) have shown that better performance can be obtained by using smaller pooling cardinalities. In the random pyramid setting, the performance of max pooling is intermediate between that obtained with whole-image and spatial pyramid pooling, while the classification using average pooling becomes worse than with whole-image pooling. A number of concurrent factors could explain the increased accuracy: (1) smaller pooling cardinality, (2) smoothing over multiple estimates (one per finer cell of the pyramid), (3) estimation of two distinct features (the maximum over the full and partial cardinalities, respectively). The more comprehensive experiments presented in the next section resolve this ambiguity by isolating each factor.

4.3.2 Experiments with binary features

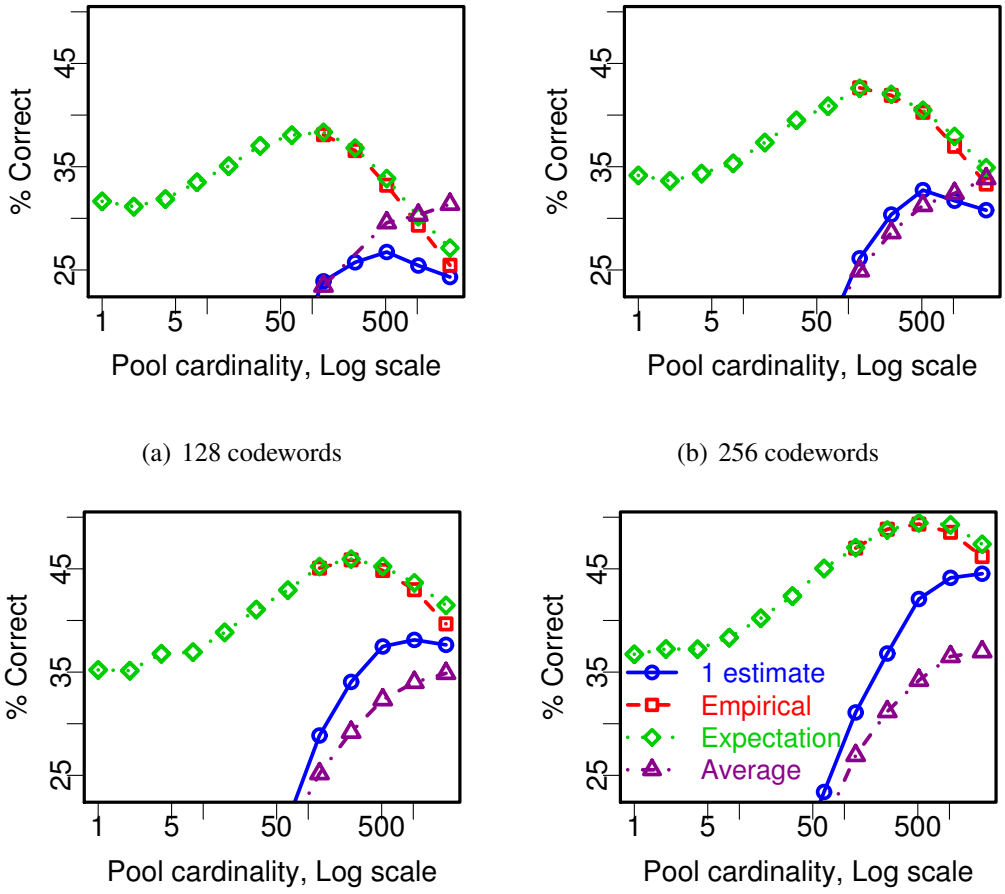
We test our conjectures by running experiments on the Scenes (Lazebnik et al., 2006) and Caltech-101 (Fei-Fei et al., 2004) datasets, which respectively comprise 101 object categories (plus a "background" category) and fifteen scene categories. In all experiments, the features being pooled are local codes representing 16×16 SIFT descriptors that have been densely extracted using the parameters yielding the best accuracy in the previous chapter (every 8 pixels for the Scenes and every 4 pixels for Caltech-101). The codes jointly represent 2×2 neighborhoods of SIFT descriptors, with macrofeature subsampling stride of 8 and 16 pixels for the Scenes and Caltech-101, respectively. Features

are pooled over the whole image using either average or max pooling. In this chapter, classification is performed with a one-versus-one support vector machine (SVM) using a linear kernel, except when otherwise stated. 100 and 30 training images per class are used for the Scenes and Caltech-101 datasets, respectively, and the rest for testing, following the usual experimental setup. We report the average per-class recognition rate, averaged over 10 random splits of training and testing images.

Optimal pooling cardinality

We first test whether recognition can indeed improve for some codebook sizes when max pooling is performed over samples of smaller cardinality, as predicted by our analysis. Recognition performance is compared using either average or max pooling, with various combinations of codebook sizes and pooling cardinalities. We use whole-image rather than pyramid or grid pooling, since having several cells of same cardinality provides some smoothing that is hard to quantify. Results are presented in Fig. (4.2) and Fig. (4.3), and show that:

- Recognition performance of average-pooled features (*Average* in the figures) increases with pooling cardinality for all codebook sizes, as expected
- performance also increases with max pooling (*1 estimate* in the figures) when the codebook size is large
- noticeable improvements appear at intermediate cardinalities for the smaller codebook sizes (compare top blue, solid curves to bottom ones), as predicted by our analysis.



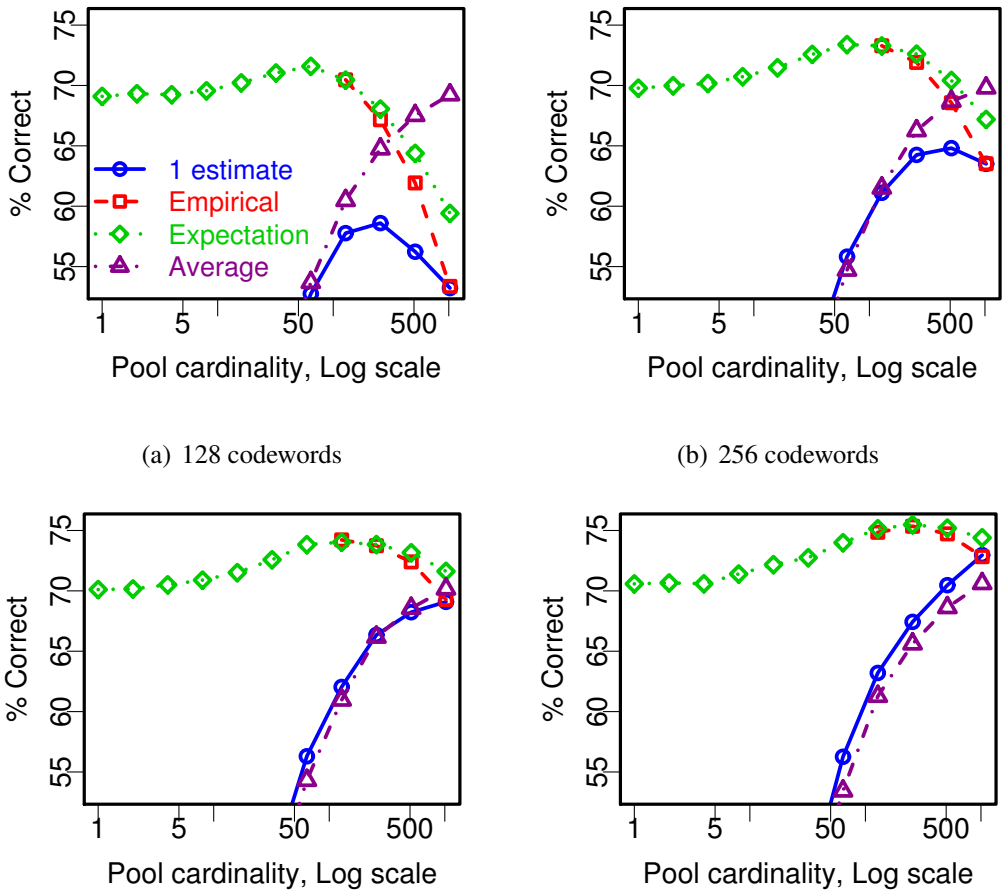
(a) 128 codewords

(b) 256 codewords

(c) 512 codewords

(d) 1024 codewords

Figure 4.2: Influence of pooling cardinality and smoothing on performance, on the Caltech-101 dataset. 1 estimate: max computed over a single pool. Empirical: empirical average of max-pooled features over several subsamples (not plotted for smaller sizes, when it reaches the expectation) Expectation: theoretical expectation of the maximum over P samples $1-(1-\xi)^P$, computed from the empirical average ξ. Average: estimate of the average computed over a single pool. Best viewed in color.



(a) 128 codewords

(b) 256 codewords

(c) 512 codewords

(d) 1024 codewords

Figure 4.3: Influence of pooling cardinality and smoothing on performance, on the Scenes dataset. 1 estimate: max computed over a single pool. Empirical: empirical average of max-pooled features over several subsamples (not plotted for smaller sizes, when it reaches the expectation) Expectation: theoretical expectation of the maximum over P samples $1 - (1 - \xi)^P$, computed from the empirical average ξ. Average: estimate of the average computed over a single pool. Best viewed in color.

Next, we examine whether better recognition can be achieved when using a smoother estimate of the expected max-pooled feature activation. We consider two ways of refining the estimate. First, if only a fraction of all samples is used, a smoother estimate can be obtained by replacing the single max by an empirical average of the max over different subsamples. Average pooling is the limit case as pool cardinality decreases. The second approach directly applies the formula for the expectation of the maximum ($1-(1-\xi)^P$, using the same notation as before) to the empirical mean computed using all samples. This has the benefit of removing the constraint that P be smaller than the number of available samples, in addition to being computationally very simple. Results using these two smoothing strategies are plotted in Fig. (4.2) and Fig. (4.3) under labels *Empirical* and *Expectation*, respectively.

Several conclusions can be drawn:

- Smoothing the estimate of the max-pooled features always helps, especially at smaller pooling cardinalities.

- The best performance is then obtained with pooling cardinalities smaller than the full cardinality in all our experiments.

- As predicted, the maximum of the curve shifts towards larger cardinality as codebook size increases.

- The best estimate of the max-pooled feature is the expectation computed from the empirical mean, $1-(1-\xi)^P$. P here simply becomes the parameter of a nonlinear function applied to the mean.

- In all cases tested, using this nonlinear function with the optimal P outperforms both average and max pooling.

Combining multiple pooling cardinalities

The maximum over a pool of smaller cardinality is not merely an estimator of the maximum over a large pool; therefore, using different pool cardinalities (e.g., using a spatial pyramid instead of a grid) may provide a more powerful feature, independently of the difference in spatial structure. Using a codebook of size 256, we compare recognition rates using jointly either one, two, or three different pooling cardinalities, with average pooling, max pooling with a single estimate per pooling cardinality, or max pooling smoothed by using the theoretical expectation. Results presented in Table (4.2) show that combining cardinalities does improve performance with max pooling — i.e., results are better for *Joint* than for all present cardinalities by themselves (*One*) — but only if the estimate has not been smoothed — i.e., when using the smooth estimate *SM*, the best cardinality by itself (*One*) is better than *Joint*. Thus, the simultaneous presence of multiple cardinalities does not seem to provide any benefit beyond that of an approximate smoothing.

Practical consequences

In papers using a spatial pyramid (Lazebnik et al., 2006; Yang et al., 2009b), there is a coupling between the pooling cardinality and other parameters of the experiment: the pooling cardinality is the density at which the underlying low-level feature representation have been extracted (e.g., SIFT features computed every 8 pixels in (Lazebnik et al., 2006)) multiplied by the spatial area of each spatial pool. While using all available sam-

Smallest cardinality		1024	512	256
Caltech 101	Avg, One	$\mathbf{32.4 \pm 1.1}$	31.3 ± 1.0	28.6 ± 1.1
	Avg, Joint		31.9 ± 1.2	32.1 ± 1.2
	Max, One	31.7 ± 1.4	32.7 ± 1.3	30.4 ± 2.3
	Max, Joint		34.4 ± 0.7	$\mathbf{35.8 \pm 0.9}$
	SM, One	37.9 ± 0.6	40.5 ± 0.7	$\mathbf{42.0 \pm 1.4}$
	SM, Joint		39.4 ± 1.3	40.6 ± 0.8
15 Scenes	Avg, One	$\mathbf{69.8 \pm 0.7}$	68.7 ± 0.8	66.3 ± 0.7
	Avg, Joint		69.6 ± 0.7	69.2 ± 1.0
	Max, One	63.5 ± 0.6	64.8 ± 0.7	64.3 ± 0.4
	Max, Joint		65.4 ± 0.6	$\mathbf{67.1 \pm 0.6}$
	SM, One	67.2 ± 0.8	70.4 ± 0.7	$\mathbf{72.6 \pm 0.7}$
	SM, Joint		69.2 ± 0.7	70.7 ± 0.7

Table 4.2: Classification results with whole-image pooling over binary codes ($k = 256$). *One* indicates that features are pooled using a single cardinality, *Joint* that the larger cardinalities are also used. *SM*: smooth maximum $(1 - (1 - \xi)^P)$.

Codebook size		256	512	1024
Caltech 101	Max	67.5 ± 1.0	69.2 ± 1.1	71.0 ± 0.8
	SM	$\mathbf{68.6 \pm 0.9}$	$\mathbf{70.0 \pm 1.2}$	$\mathbf{71.8 \pm 0.8}$
15 Scenes	Max	77.9 ± 0.7	79.4 ± 0.5	80.2 ± 0.4
	SM	$\mathbf{78.2 \pm 0.4}$	$\mathbf{79.9 \pm 0.5}$	$\mathbf{80.5 \pm 0.6}$

Table 4.3: Recognition accuracy with 3-level pyramid pooling over binary codes. One-vs-all classification has been used in this experiment. *Max*: max pooling using all samples. *SM*: smooth maximum (expected value of the maximum computed from the average $1 - (1 - \xi)^P$), using a pooling cardinality of $P = 256$ for codebook sizes 256 and 512, $P = 512$ for codebook size 1024.

ples is optimal for average pooling, this is usually not the case with max pooling over binary features, particularly when the size of the codebook is small. Instead, the pooling cardinality for max pooling should be adapted to the dictionary size, and the remaining samples should be used to smooth the estimate. Another, simpler way to achieve similar or better performance is to apply to the average-pooled feature the nonlinear transformation corresponding to the expectation of the maximum, (i.e., $1-(1-\xi)^P$, using the same notation as before); in addition, the parameter P is then no longer limited by the number of available samples in a pool, which may be important for very large codebooks. Our experiments using binary features in a three-level pyramid show that this transformation yields improvement over max pooling for all codebook sizes (Table (4.3)). The increase in accuracy is small, however the difference is consistently positive when looking at experimental runs individually instead of the difference in the averages over ten runs.

4.3.3 Pooling continuous sparse codes

Sparse codes have proven useful in many image applications such as image compression and deblurring. Combined with max pooling, they have led to state-of-the-art image recognition performance with a linear classifier (Yang et al., 2009b; Boureau et al., 2010a). However, the analysis developed for binary features in the previous section does not apply, and the underlying causes for this good performance seem to be different.

Influence of pooling cardinality

In the case of binary features, and when no smoothing is performed, we have seen above that there is an optimal pooling cardinality, which increases with the sparsity of the features. Smoothing the features displaces that optimum towards smaller cardinalities. In this section, we perform the same analysis for continuous features, and show that (1) it is always better to use all samples for max pooling when no smoothing is performed, (2) however the increase in signal-to-noise ratio (between means' separation and standard deviation) does not match the noise reduction obtained by averaging over all samples.

Model

Let P denote cardinality of the pool. Exponential distribution (or Laplace distributions for feature values that may be negative) are often preferred to Gaussian distributions to model visual feature responses because they are highly kurtotic. In particular, they are a better model for sparse codes. Assume the distribution of the value of a feature for each patch is an exponential distribution with mean $1/\lambda$ and variance $1/\lambda^2$. The corresponding cumulative distribution function is $1 - e^{-\lambda x}$. The cumulative distribution function of the max-pooled feature is $(1 - e^{-\lambda x})^P$. The mean and variance of the distri-

bution of the max-pooled feature can be shown[2] to be respectively $\mu_m = \mathcal{H}(P)/\lambda$ and $\sigma_m^2 = 1/\lambda^2 \sum_{l=1}^{P} 1/l(2\mathcal{H}(l) - \mathcal{H}(P))$, where $\mathcal{H}(k) = \sum_{i=1}^{k} 1/i$ denotes the harmonic series. Thus, for all P, $\mu_1/\mu_2 = \sigma_1/\sigma_2 = \lambda_1/\lambda_2$, and the distributions will be better separated if the scaling factor of the mean is bigger than the scaling factor of the standard deviations, i.e., $\mathcal{H}(P) > \sqrt{\sum_{l=1}^{P} 1/l(2\mathcal{H}(l) - \mathcal{H}(P))}$, which is true for all P. Furthermore, since $\mathcal{H}(P) = \log(P) + \gamma + o(1)$ when $P \to \infty$ (where γ is Euler's constant), it can be shown that $\sum_{l=1}^{P} 1/l(2\mathcal{H}(l) - \mathcal{H}(P)) = \log(P) + O(1)$, so that the distance between the means grows faster (like $\log(P)$) than the standard deviation, which grows like $\sqrt{\log(P)}$. Two conclusions can be drawn from this: (1) when no smoothing is performed, larger cardinalities provide a better signal-to-noise ratio, but (2) this ratio grows slower than when simply using the additional samples to smooth the estimate ($1/\sqrt{P}$

[2] PROOF:

- MEAN: The cumulative distribution function of the max-pooled feature is $F(x) \triangleq (1 - \exp(-\lambda x))^P$. Hence,

$$\mu_m = \int_0^\infty [1 - F(x)]dx = \int_0^\infty 1 - (1 - \exp(-\lambda x))^P dx.$$

Changing variable to $u = (1 - \exp(-\lambda x))$:

$$\mu_m = \frac{1}{\lambda} \int_0^1 \frac{1 - u^P}{1 - u} du = \frac{1}{\lambda} f(P),$$

where $f_1(P) \triangleq \int_0^1 \frac{1-u^P}{1-u} du$. We have:

$$f_1(0) = 0; \forall P \geq 1, f_1(P) - f_1(P-1) = \int_0^1 u^{P-1} du = \frac{1}{P}.$$

Hence, $\mu_m = 1/\lambda \sum_{j=1}^{P} 1/j = \mathcal{H}(P)/\lambda$. □

- VARIANCE: see appendix.

assuming independent samples, although in reality smoothing is less favorable since the independence assumption is clearly false in images).

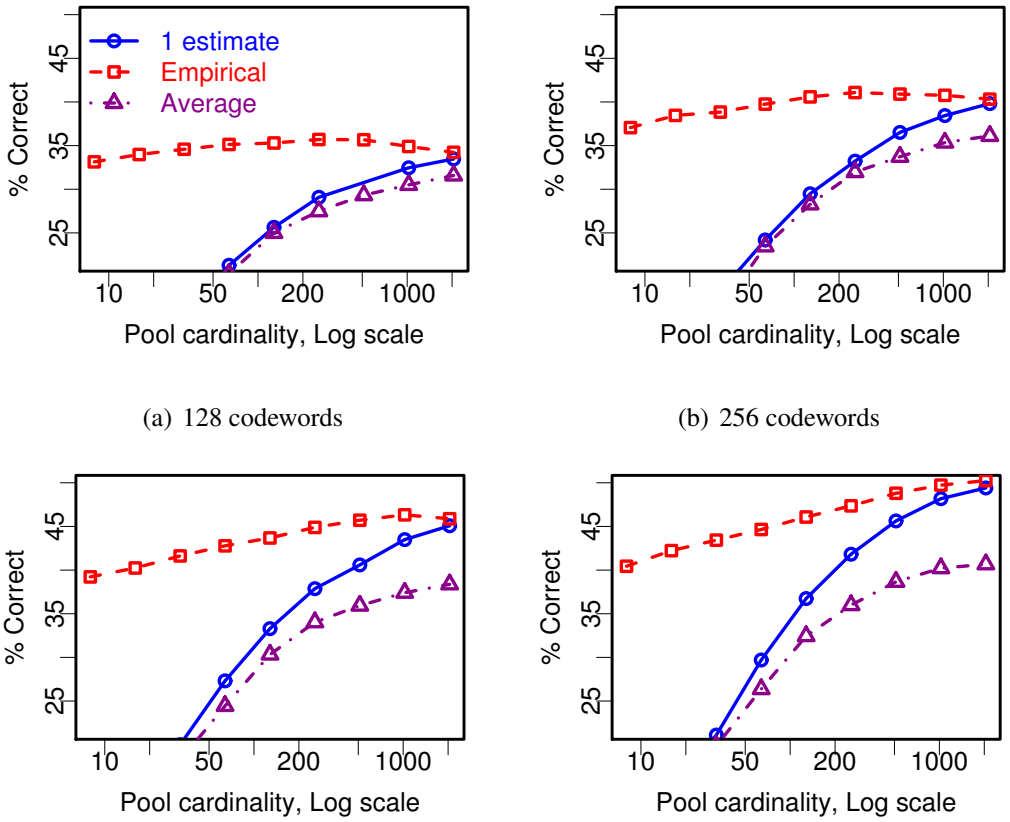
4.3.4 Experiments with sparse features

We perform the same experiments as in the previous section to test the influence of codebook size and pooling cardinalities, using continuous sparse codes instead of binary codes.

Results are presented in Fig. (4.4) and Fig. (4.5). As expected from our analysis, using larger pooling cardinalities is always better with continuous codes when no smoothing is performed (blue solid curve): no bump is observed even with smaller dictionaries. Max pooling performs better than average pooling on the Caltech dataset (Fig. (4.4)); this is not predicted by the analysis using our very simple model. On the Scenes dataset (Fig. (4.5)), max pooling and average pooling perform equally well when the largest dictionary size tested (1024) is used. Slightly smoothing the estimate of max pooling by using a smaller sample cardinality results in a small improvement in performance; since the grid (or pyramid) pooling structure performs some smoothing (by providing several estimates for the sample cardinalities of the finer levels), this may explain part of the better performance of max pooling compared to average pooling with grid and pyramid smoothing, even though average pooling may perform as well when a single estimate is given.

Combining pooling cardinalities

Our analysis predicts that combining several cardinalities should not result in drastically improved performance. Results in Table (4.4) indeed show very limited or no improve-



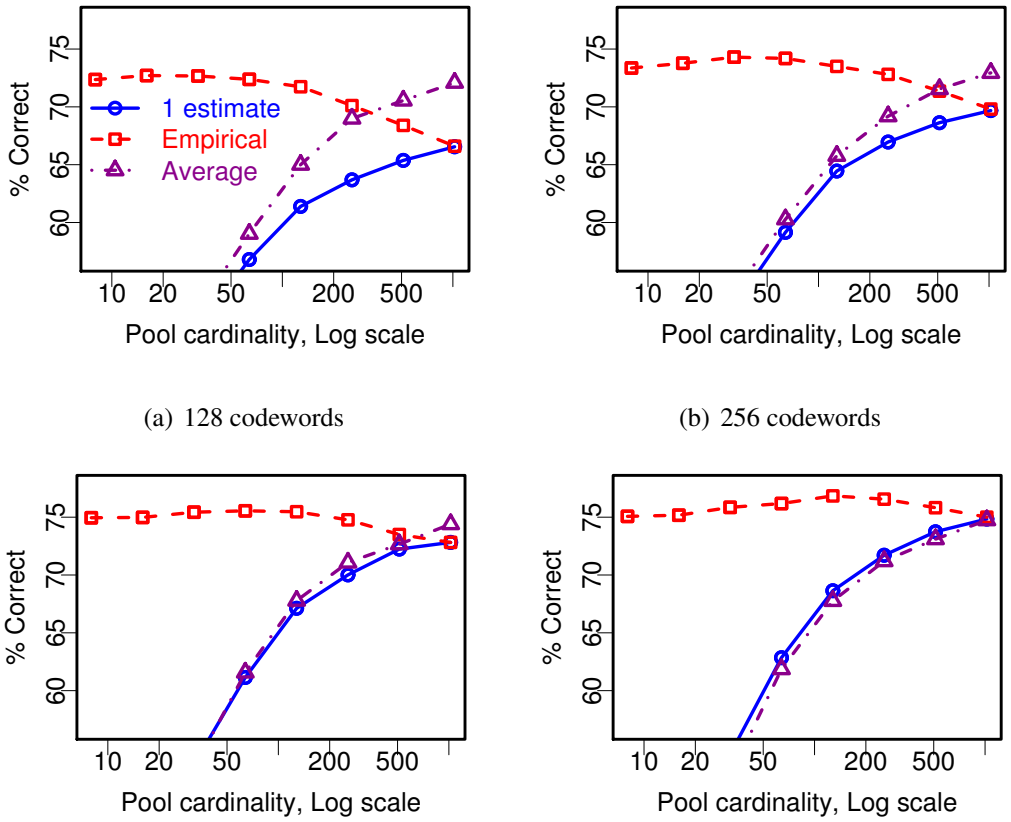
(a) 128 codewords

(b) 256 codewords

(c) 512 codewords

(d) 1024 codewords

Figure 4.4: Influence of pooling cardinality and smoothing on performance, on the Caltech-101 dataset. 1 estimate: maximum computed over a single pool. Empirical: empirical average of max-pooled features over several subsamples of smaller cardinality. Average: estimate of the average computed over a single pool. Best viewed in color.



(a) 128 codewords

(b) 256 codewords

(c) 512 codewords

(d) 1024 codewords

Figure 4.5: Influence of pooling cardinality and smoothing on performance, on the Scenes dataset. 1 estimate: maximum computed over a single pool. Empirical: empirical average of max-pooled features over several subsamples of smaller cardinality. Average: estimate of the average computed over a single pool. Best viewed in color.

Smallest cardinality		1024	512	256
Caltech 101	Avg, One	$\mathbf{35.4 \pm 1.7}$	33.7 ± 1.6	32.0 ± 0.9
	Avg, Joint		35.1 ± 1.4	34.6 ± 1.2
	Max, One	$\mathbf{38.4 \pm 1.0}$	36.5 ± 1.5	33.2 ± 0.8
	Max, Joint		$\mathbf{38.4 \pm 1.8}$	37.5 ± 1.8
15 Scenes	Avg, One	$\mathbf{72.9 \pm 0.7}$	71.5 ± 0.8	69.2 ± 0.7
	Avg, Joint		72.6 ± 0.7	71.9 ± 0.7
	Max, One	69.7 ± 0.8	68.6 ± 0.9	67.0 ± 0.6
	Max, Joint		69.2 ± 2.0	$\mathbf{70.3 \pm 0.4}$

Table 4.4: Classification results with whole-image pooling over sparse codes ($k = 256$). *One* indicates that features are pooled using a single cardinality, *Joint* that the larger cardinalities are also used. Here, using several cardinalities does not increase accuracy with either average or max pooling.

ment when pooling cardinalities are combined.

4.4 Mixture distribution and clutter model

Our simple model does not account for the better discrimination sometimes achieved by max pooling for continuous sparse codes with large dictionaries. In practice, the ideal case of all data points coming from one of two classes is rarely encountered. We briefly present how max pooling may also help in a slightly more realistic case. When doing visual recognition, patches that are highly specific to a class are found alongside

generic patches that contribute little discrimination information. These can be background patches, or plain patches that are pervasive in all classes. The fraction of informative patches may vary widely between images, making statistical inference harder.

With the same notation as before, consider a binary linear classification task over cluttered images. Pooling is performed over the whole image, so that the pooled feature $\boldsymbol{h}$ is the global image representation. Linear classification requires distributions of $\boldsymbol{h}$ over examples from positive and negative classes (henceforth denoted by $+$ and $-$) to be well separated.

We model the distribution of image patches of a given class as a mixture of two distributions (Minka, 2001): patches are taken from the actual class distribution (foreground) with probability $(1 - w)$, and from a clutter distribution (background) with probability w, with clutter patches being present in both classes ($+$ or $-$). Crucially, we model the amount of clutter w as varying between images (while being fixed for a given image).

There are then two sources of variance for the distribution $p(\boldsymbol{h})$: the intrinsic variance caused by sampling from a finite pool for each image (which causes the actual value of $\boldsymbol{h}$ over foreground patches to deviate from its expectation), and the variance of w (which causes the expectation of $\boldsymbol{h}$ itself to fluctuate from image to image depending on their clutter level). If the pool cardinality N is large, average pooling is robust to intrinsic foreground variability, since the variance of the average decreases like $1/N$. This is usually not the case with max pooling, where the variance can increase with pool cardinality depending on the foreground distribution.

However, if the amount of clutter w has a high variance, it causes the distribution of

the average over the image to spread, since the expectation of $\boldsymbol{h}$ for each image depends on w. Even if the foreground distributions are well separated, variance in the amount of clutter creates overlap between the mixture distributions if the mean of the background distribution is much lower than that of the foreground distributions. Conversely, max pooling can be robust to clutter if the mean of the background distribution is sufficiently low. This is illustrated in Fig. (4.6), where we have plotted the empirical distributions of the average of 10 pooled features sharing the same parameters. Simulations are run using 1000 images of each class, composed of $N = 500$ patches. For each image, the clutter level w is drawn from a truncated normal distribution with either low (top) or high (bottom) variance. Local feature values at each patch are drawn from a mixture of exponential distributions, with a lower mean for background patches than foreground patches of either class. When the clutter has high variance (Fig. (4.6), bottom), distributions remain well separated with max pooling, but have significant overlap with average pooling.

We now refine our analysis in two cases: sparse codes and vector quantized codes.

Sparse codes.

In the case of a positive decomposition over a dictionary, as before, we model the distribution of the value of feature j for each patch by an exponential distribution with mean μ_j, variance μ_j^2, and density $f(x) = 1/\mu_j \exp(-x/\mu_j)$.

The corresponding cumulative distribution function is $F(x) = 1 - \exp(-x/\mu_j)$. The cumulative distribution function of the max-pooled feature with a pool of size P is $F^P(x) = (1 - \exp(-x/\mu_j))^P$. Clutter patches are sampled from a distribution of mean μ_b. Let P_f and P_b denote respectively the number of foreground and background patches,

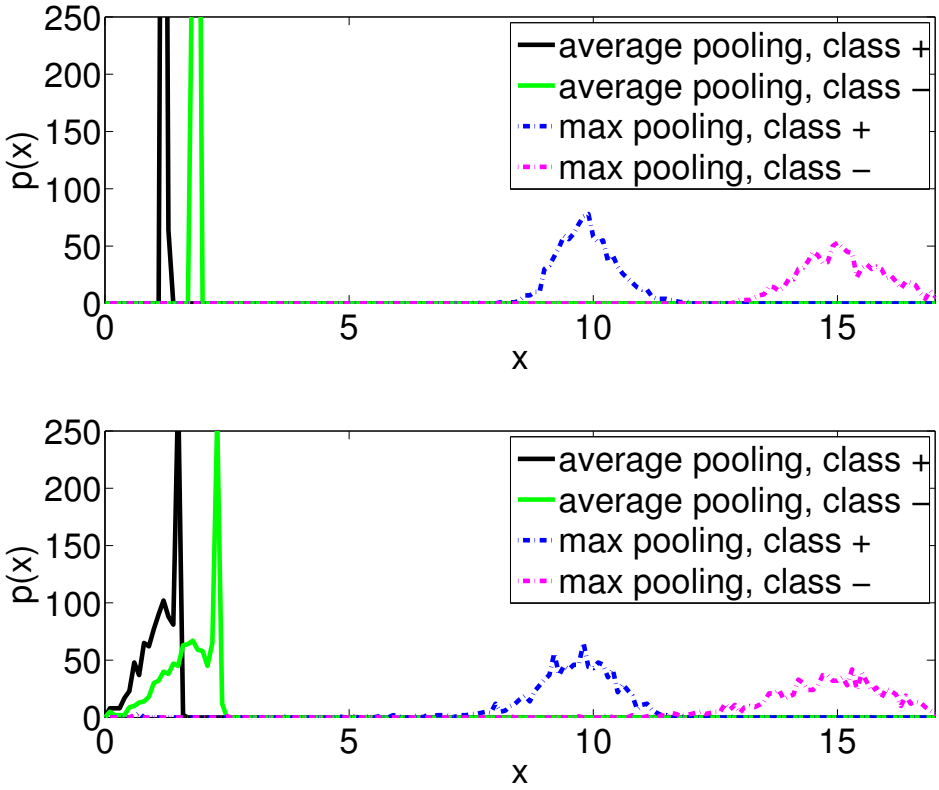


Figure 4.6: Empirical probability densities of $x = \frac{1}{K}\sum_{j=1}^{K} h_j$, simulated for two classes classes of images forming pools of cardinality $N = 500$. The local features are drawn from one of three exponential distributions. When the clutter is homogeneous across images (top), the distributions are well separated for average pooling and max pooling. When the clutter level has higher variance (bottom), the max pooling distributions (dashed lines) are still well separated while the average pooling distributions (solid lines) start overlapping.

$P = P_f + P_b$. Assuming P_f and P_b are large, Taylor expansions of the cumulative distribution functions of the maxima yield that 95% of the probability mass of the maximum over the background patches will be below 95% of the probability mass of the maximum over the foreground patches provided that $P_b < |\log(0.95)| \, (P_f/|\log(0.05)|)^{\mu_j/\mu_b}$. In a binary discrimination task between two comparatively similar classes, if an image is cluttered by many background patches, with $\mu_b \ll \mu_j^+$ and $\mu_b \ll \mu_j^-$, max-pooling can be relatively immune to background patches, while average-pooling can create overlap between the distributions (see Fig. (4.6)). For example, if $\mu_b < 2\mu_j$ and $P_f = 500$, having fewer than $P_b < 1400$ background patches virtually guarantees that the clutter will have no influence on the value of the maximum. Conversely, if $P_b < P_f/59 \leq |\log(0.95)|/|\log(0.05)|P_f$, clutter will have little influence for μ_b up to μ_j. Thus, max-pooling creates immunity to two different types of clutter: ubiquitous with low feature activation, and infrequent with higher activation.

Vector quantization.

We model binary patch codes for a given feature as before, as i.i.d. Bernoulli random variables of mean ξ. The distribution of the average-pooled feature also has mean ξ, and its variance decreases like $1/P$. The maximum is a Bernoulli variable of mean $1-(1-\xi)^P$ and variance $(1-(1-\xi)^P)(1-\xi)^P$. Thus, it is 1 with probability 0.95 if:
$P \geq \log(0.05)/\log(1-\xi) \approx |\log(0.05)|/\xi$, and 0 with probability 0.95 if:
$P \leq \log(0.95)/\log(1-\xi) \approx |\log(0.95)|/\xi$, for $\xi \ll 1$. The separability of classes depends on sample cardinality P. There exists a sample cardinality P for which the maximum over class $+$ is 0 with probability 0.95, while the maximum over class $-$ is 1

with probability 0.95, if:

$$\frac{\xi^-}{\xi^+} > \frac{\log(0.05)}{\log(0.95)}, \text{ e.g. if } \frac{\xi^-}{\xi^+} > 59.$$

Since $\sum_j \xi = 1$ in the context of vector quantization, ξ becomes very small on average if the codebook is very large. For $\xi \ll 1$, the characteristic scale of the transition from 0 to 1 is $1/\xi$, hence the pooling cardinality range corresponding to easily separable distributions can be quite large if the mean over foreground patches from one class is much higher than both the mean over foreground patches from the other class and the mean over background patches.

4.5 Transition from average to max pooling

The previous sections have shown that depending on the data and features, either max or average pooling may perform best. The optimal pooling type for a given classification problem may be neither max nor average pooling, but something in between; in fact, we have shown that it is often better to take the max over a fraction of all available feature points, rather than over the whole sample. This can be viewed as an intermediate position in a parametrization from average pooling to max pooling over a sample of fixed size, where the parameter is the number of feature points over which the max is computed: the expected value of the max computed over one feature is the average, while the max computed over the whole sample is obviously the real max.

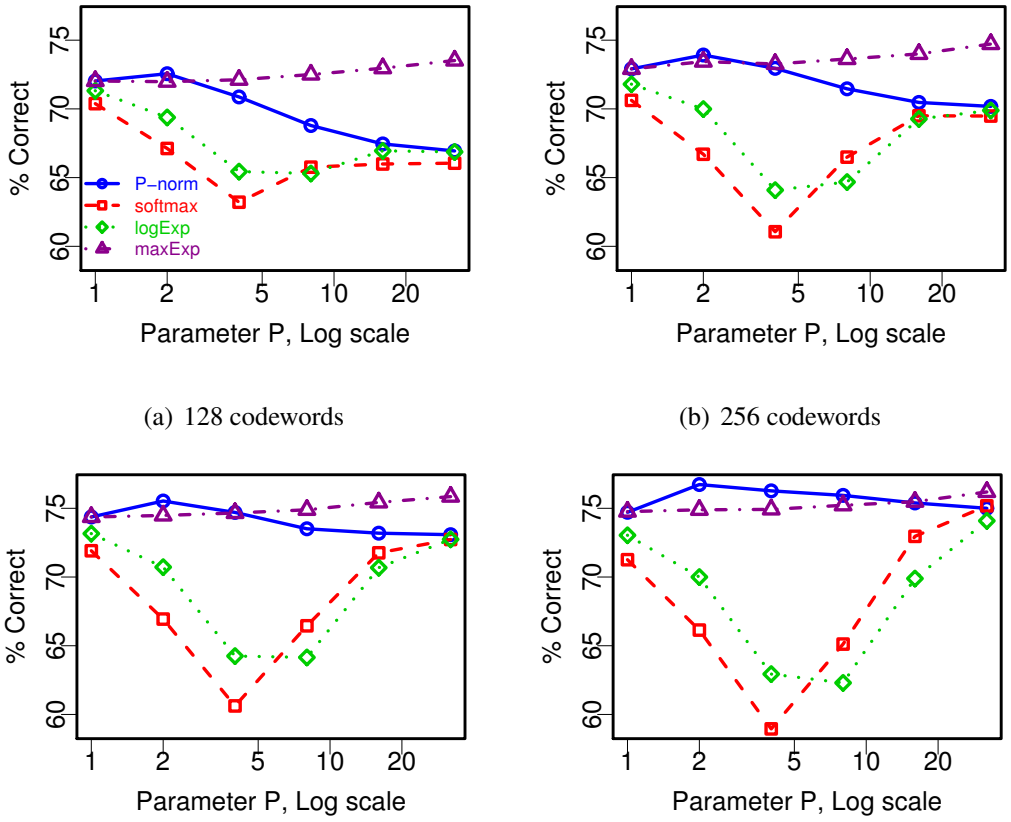
This is only one of several possible parametrizations that continuously transition from average to max pooling. The P-norm of a vector (more accurately, a version of it normalized by the number of samples N) is another well-known one: $f_P(\boldsymbol{v}) =$

$\left(\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{v}_i^P\right)^{\frac{1}{P}}$, which gives the average for $P = 1$ and the max for $P \to \infty$. This parametrization accommodates ℓ_2-norm pooling, which is also square-root pooling when features are binary (for $P = 2$), and absolute value pooling (for $P = 1$), both of which have been used in the literature (e.g., (Yang et al., 2009b)). In our experiments, ℓ_2-pooling appears to be the best choice for P-norm pooling.

A third parametrization is the sum of samples weighted by a softmax function: $\sum_i \exp(\beta \boldsymbol{x}_i)/\sum_j \exp(\beta \boldsymbol{x}_j)\boldsymbol{x}_i$. This gives average pooling for $\beta = 0$ and max pooling for $\beta \to \infty$. Finally, a fourth parametrization is $\frac{1}{\beta}\log\frac{1}{N}\sum_i \exp(\beta \boldsymbol{x}_i)$, which gives the average for $\beta \to 0$ and the max for $\beta \to \infty$. As with the P-norm, the result only depends on the empirical feature activation mean in the case of binary vectors; thus, these functions can be applied to an already obtained average pool.

Fig. (4.7) plots the recognition rate obtained on the Scenes dataset using sparse codes and each of the four parametrizations mentioned. Instead of using the expectation of the maximum for exponential distributions, we have used the expectation of the maximum of binary codes ($1 - (1 - \xi)^P$), applied to the average, as we have observed that it works well; we refer to this function as the expectation of the maximum (*maxExp* in Fig. (4.7)), although it does not converge to the maximum when $P \to \infty$ for continuous codes. Both this parametrization and the P-norm perform better than the two other pooling functions tested, which present a marked dip in performance for intermediate values.

Fig. (4.8) shows the expected value of pooled features according to our model for binary features, for the same parametrizations. Separation is always better achieved when features are rare (top row) than when they are often active (bottom row).



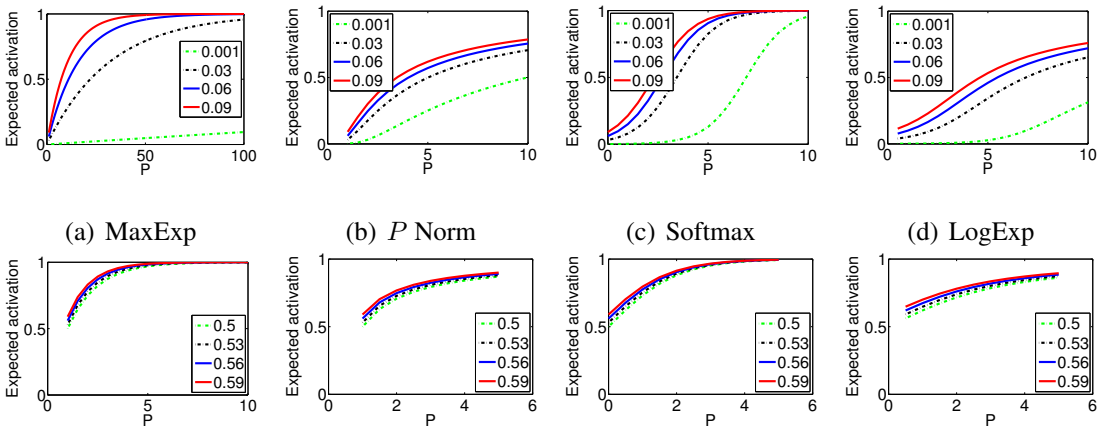
(a) 128 codewords

(b) 256 codewords

(c) 512 codewords

(d) 1024 codewords

Figure 4.7: Recognition rate obtained on the scenes dataset using several pooling functions that perform a continuous transition from average to max pooling when varying parameter P (see text). Best viewed in color.



(a) MaxExp (b) P Norm (c) Softmax (d) LogExp

(e) (f) (g) (h)

Figure 4.8: Several continuous parametrizations from average to max. For all parametrizations, separation can be increased when average feature activation is small (upper row), but the transformation is not very useful with larger activations. The legend gives the values of ξ used for plotting.

4.6 Conclusion

This chapter has shown that the ability of the pooled feature to discriminate between classes crucially depends on the statistics of the local features to be pooled, and the composition of the sample over which pooling is performed. If the pooling step is fixed, then some coding steps may be more suited to the particular type of information crushing exerted by the pooling step. In particular, max pooling has good discrimination properties when the features being pooled are rare. This may partly explain why sparse features are well-suited to max pooling.

5
LOCALITY IN CONFIGURATION SPACE

The previous chapter has looked at how best to represent the information within a given pool, taking the sample within a pool as fixed. This chapter now turns to the choice of the pools themselves. Previous work has shown that simply making the pools more spatially restricted (e.g., the cells of a grid instead of the whole image) makes the representation more powerful. Here, we look at the effect of restricting pooling to vectors that are similar to one another; i.e., taking into account locality in configuration space to draw the neighborhoods. The research presented in this chapter has been published in (Boureau et al., 2011).

5.1 Introduction

Much recent work in image recognition has underscored the importance of locality constraints for extracting good image representations. Methods that incorporate some way of taking locality into account define the state of the art on many challenging image classification benchmarks such as Pascal VOC, Caltech-101, Caltech-256, and 15-Scenes (Gao et al., 2010; Wang et al., 2010; Yang et al., 2010; Yu et al., 2009; Zhou et al., 2010).

While the pooling operations are often performed over local spatial neighborhoods, the neighborhoods may contain feature vectors that are very heterogeneous, possibly leading to the loss of a large amount of information about the distribution of features, as illustrated in Fig. (5.1). Restricting the pooling to feature vectors that are similar in the

multidimensional input space (or nearby) (Jégou et al., 2010; Zhou et al., 2010) remedies this problem. Considering similar inputs for smoothing noisy data over a homogeneous sample reduces noise without throwing out the signal, and has long been recognized useful in the image processing and denoising communities (Buades et al., 2005; Dabov et al., 2006). This trick has been successfully incorporated to denoising methods using sparse coding (Mairal et al., 2009b). It is interesting to note that considerations of locality often pull coding and pooling in opposite directions: they make coding smoother (neighbors are used to regularize coding so that noise is harder to represent) and pooling more restrictive (only neighbors are used so that the signal does not get averaged out). This can be viewed as an attempt to distribute smoothing more evenly between coding and pooling.

Authors of locality-preserving methods have often attributed their good results to the fact that the encoding uses only dictionary atoms that resemble the input (Wang et al., 2010; Yu et al., 2009), or viewed them as a trick to learn huge specialized dictionaries, whose computational cost would be prohibitive with standard sparse coding (Yang et al., 2010). The locality that matters in these methods is stated to be locality between *inputs and atoms*, not preservation of locality *across inputs* so that similar inputs have similar codes. However, the triangle inequality implies that locality across inputs is always somewhat preserved if codes use atoms that are close to inputs, but the reverse is not true. Thus, focusing on similarity between inputs and atoms may underestimate the influence of the preservation of locality. We argue that more local pooling may be one factor in the success of methods that incorporate locality constraints into the training criterion of the codebook for sparse coding (Gao et al., 2010; Wang et al., 2010; Yu et al., 2009),

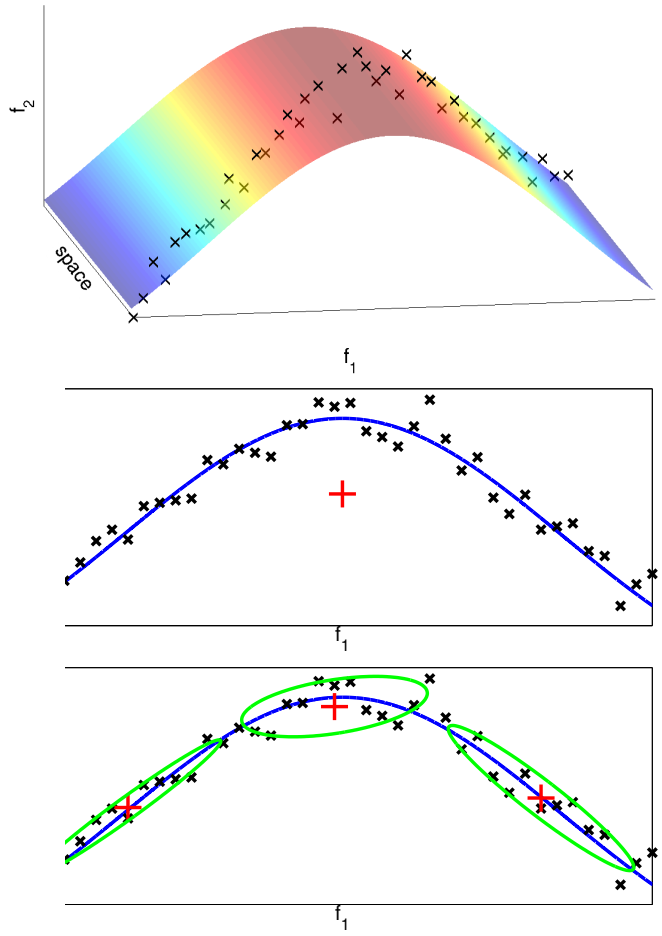


Figure 5.1: Cartoon representation of a distribution of descriptors that has a high curvature and is invariant to the spatial location in the image, with two feature components (top). The middle and bottom figures show the samples projected across space in the 2D feature space. Due to the curvature of the surface, global pooling (middle) loses most of the information contained in the descriptors; the red cross (average pooling of the samples) is far away from the lower-dimensional surface on which the samples lie. Clustering the samples and performing pooling inside each cluster preserves information since the surface is locally flat (bottom).

or directly cluster the input data to learn one local dictionary per cluster (Wang et al., 2010; Yang et al., 2010). The question we attempt to answer in this chapter is whether it is possible to leverage locality in the descriptor space once the descriptors have already been encoded. We argue that if the coding step has not been designed in such a way that the pooling operation preserves as much information as possible about the distribution of features, then *the pooling step itself should become more selective*.

The contributions of this work are threefold. First, we show how several recent feature extracting methods can be viewed in a unified perspective as preventing pooling from losing too much relevant information. Second, we demonstrate empirically that restricting pools to codes that are nearby not only in (2D) image space but also in descriptor space, boosts the performance even with relatively small dictionaries, yielding state-of-the-art performance or better on several benchmarks, without resorting to more complicated and expensive coding methods, or having to learn new dictionaries. Third, we propose some promising extensions.

5.2 Pooling more locally across the input space

We propose to streamline the approach in (Yang et al., 2010), which requires learning one different dictionary per cluster, and show that simply making the pooling step more selective can substantially enhance the performance of small dictionaries, and beat the state of the art on some object recognition benchmarks when large dictionaries are used, without requiring additional learning beyond obtaining an additional clustering codebook with K-means. Comparing the performance of our system with that obtained with individual dictionaries allows us to quantify the relative contributions of more selective

pooling and more specialized, overcomplete dictionaries.

To clarify how our local pooling scheme differs from the usual local spatial pooling, an image feature can be viewed as a couple $z = (\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{y} \in \mathbb{R}^2$ denotes a pixel location, and $\boldsymbol{x} \in \mathbb{R}^d$ is a vector, or configuration, encoding the local image structure at $\boldsymbol{y}$ (e.g., a SIFT descriptor, with $d = 128$). A feature set $\mathcal{Z}$ is associated with each image, its size potentially varying from one picture to the next.

Spatial pooling considers a fixed – that is, predetermined and image-independent – set of M possibly overlapping image regions (spatial bins) $\mathcal{Y}_1$ to $\mathcal{Y}_M$. To these, we add a fixed set of P (multi-dimensional) bins $\mathcal{X}_1$ to $\mathcal{X}_P$ in the configuration space. In this work, the spatial bins are the cells in a spatial pyramid, and the configuration space bins are the Voronoi cells of clusters obtained using K-means.

Denoting by g the pooling operator (average or max in the previous section), the pooled feature is obtained as:

$$\boldsymbol{h}_{(m,p)} = g_{(i \in \mathcal{Y}_m, j \in \mathcal{X}_p)}(\boldsymbol{\alpha}_{(i,j)}). \tag{5.1}$$

Bags of features can be viewed as a special case of this in two ways: either by considering the 1-of-K encoding presented above, followed by global pooling in the configuration space ($P = 1$), or with a simplistic encoding that maps all inputs to 1, but does fine configuration space binning ($P = K$). Accordingly, the feature extraction in this chapter can be viewed either as extending the sparse coding spatial pyramid by making configuration space pooling local, or as extending the hard-vector-quantized spatial pyramid by replacing the simplistic code by sparse coding: descriptors are first decomposed by sparse coding over a dictionary of size K; the same descriptors are also clustered over a K-means dictionary of size P; finally, pooling of the sparse codes is

then performed separately for each cluster (as in aggregated coding (Jégou et al., 2010) – see Sec. (5.3) –, but with sparse codes), yielding a feature of size $K \times P \times S$ if there are S spatial bins.

While this does not apply to max pooling, local pooling can be viewed as implementing local bilinear classification when using average pooling and linear classification: the pooling operator and the classifier may be swapped, and classification of local features then involves computing $\boldsymbol{\beta}_{xy}^T W \boldsymbol{\alpha}$, where $\boldsymbol{\beta}_{xy}$ is a $(S \times P)$- dimensional binary vector that selects a subset of classifiers corresponding to the configuration space and spatial bins, and W is a $(S \times P) \times K$ matrix containing one K-dimensional local classifier per row.

5.3 Related work about locality in feature space

We start our review of previous work with a caveat about word choice. There exists an unfortunate divergence in the vocabulary used by different communities when it comes to naming methods leveraging neighborhood relationships in feature space: what is called "non-local" in work in the vein of signal processing (Buades et al., 2005; Mairal et al., 2009b) bears a close relationship to "local" fitting and density estimation (Gao et al., 2010; Saul and Roweis, 2003; Wang et al., 2010; Yu et al., 2009). Thus, non-local means (Buades et al., 2005) and locally-linear embedding (Saul and Roweis, 2003) actually perform the same type of initial grouping of input data by minimal Euclidean distance. This discrepancy stems from the implicit understanding of "local" as either "spatially local", or "local in translation-invariant configuration space".

5.3.1 Preserving neighborhood relationships during coding

Previous work has shown the effectiveness of preserving configuration space locality during coding, so that similar inputs lead to similar codes. This can be done by explicitly penalizing codes that differ for neighbors. The DrLIM system of siamese networks in (Hadsell et al., 2006), and neighborhood component analysis (Goldberger et al., 2004), learn a mapping that varies smoothly with some property of the input by minimizing a cost which encourages similar inputs to have similar codes (similarity can be defined arbitrarily, as locality in input space, or sharing the same illumination, orientation, etc.) Exploiting image self-similarities has also been used successfully for denoising (Buades et al., 2005; Dabov et al., 2006; Mairal et al., 2009b).

Locality constraints imposed on the coding step have been adapted to classification tasks with good results. Laplacian sparse coding (Gao et al., 2010) uses a modified sparse coding step in the spatial pyramid framework. A similarity matrix of input SIFT descriptors is obtained by computing their intersection kernel, and used in an added term to the sparse coding cost. The penalty to pay for the discrepancy between a pair of codes is proportional to the similarity of the corresponding inputs. This method obtains state-of-the-art results on several object recognition benchmarks. Locality-constrained linear coding (Wang et al., 2010) (LLC) projects each descriptor on the space formed by its k nearest neighbors (k is small, e.g., $k = 5$). This procedure corresponds to performing the first two steps of the locally linear embedding algorithm (Saul and Roweis, 2003) (LLE), except that the neighbors are selected among the atoms of a dictionary rather than actual descriptors, and the weights are used as features instead of being mere tools to learn an embedding.

Sparse coding methods incorporating a locality constraint share the property of indirectly limiting activation of a given component of the vectors representing descriptors to a certain region of the configuration space. This may play a role in their good performance. For example, in LLC coding, the component corresponding to a given dictionary atom will be non-zero only if that atom is one of the k nearest neighbors of the descriptor being encoded; the non-zero values aggregated during pooling then only come from these similar descriptors. Several approaches have implemented this strategy directly during the pooling step, and are presented in the next section.

5.3.2 Letting only neighbors vote during pooling

Pooling involves extracting an ensemble statistic from a potentially large group of inputs. However, pooling too drastically can damage performance, as shown in the spatial domain by the better performance of spatial pyramid pooling (Lazebnik et al., 2006) compared to whole-image pooling.

Different groups have converged to a procedure involving preclustering of the input to create independent bins over which to pool the data. In fact, dividing the feature space into bins to compute correspondences has been proposed earlier by the pyramid match kernel approach (Grauman and Darrell, 2005). However, newer work does not tile the feature space evenly, relying instead on unsupervised clustering techniques to adaptively produce the bins.

The methods described here all perform an initial (hard or soft) clustering to partition the training data according to appearance, as in the usual bag-of-words framework, but then assigning a vector to each cluster instead of a scalar. The representation is then a

"super-vector" that concatenates these vectors instead of being a vector that concatenates scalars.

Aggregated coding (Jégou et al., 2010) and super-vector coding (Zhou et al., 2010) both compute, for each cluster, the average difference between the inputs in the cluster, and its centroid: (1) SIFT descriptors $\mathbf{x}_i$ are extracted at regions of interest, (2) visual words $\mathbf{c}_k$ are learned over the whole data by K-means, (3) descriptors of each image are clustered, (4) for each cluster C_k, the sum $\sum_{\mathbf{x} \in C_k} (\mathbf{x} - \mathbf{c}_k)$ is computed, (5) the image descriptor is obtained by concatenating the representations for each cluster.

If the centroids were computed using only the descriptors in a query image, the representation would be all zeros, because the centroids in K-means are also obtained by averaging the descriptors in each cluster. Instead, the centroids are computed using descriptors from the whole data, implicitly representing a "baseline image" against which each query image is compared. Thus, encoding relatively to the cluster centroid removes potentially complex but non-discriminative information. This representation performs very well on retrieval (Jégou et al., 2010) and image classification (Zhou et al., 2010) (Pascal VOC2009) benchmarks.

Another related method (Yang et al., 2010) that obtains high accuracy on the Pascal datasets combines the preclustering step of aggregated and super-vector coding, with sparse decomposition over individual local dictionaries learned inside each cluster. Both approaches using preclustering for image classification (Yang et al., 2010; Zhou et al., 2010) have only reported results using gigantic global descriptors for each image. Indeed, the high results obtained in (Yang et al., 2010) are attributed to the possibility of learning a very large overcomplete dictionary (more than 250,000 atoms) which would

be computationally infeasible without preclustering, but can be done by assembling a thousand or more smaller local dictionaries. The experiments presented in the next section seek to isolate the effect of local pooling that is inherent in this scheme.

5.4 Experiments

We perform experiments on three image recognition datasets: 15-Scenes (Lazebnik et al., 2006), Caltech-101 (Fei-Fei et al., 2004) and Caltech-256 (Griffin et al., 2007). All features are extracted from grayscale images. Large images are resized to fit inside a 300×300 box. SIFT descriptors are extracted densely over the image, and encoded into sparse vectors using the SPAMS toolbox (SPAMS, 2012). We adopt the denser 2×2 macrofeatures of Chapter 3, extracted every 4 pixels, for the Caltech-256 and Caltech-101 databases, and every 8 pixels for the Scenes, except for some experiments on Caltech-256 where standard features extracted every 8 pixels are used for faster processing. The sparse codes are pooled inside the cells of a three-level pyramid (4×4, 2×2 and 1×1 grids); max pooling is used for all experiments except those in Sec. (5.4.2), which compare it to other pooling schemes. We apply an $\ell_{1.5}$ normalization to each vector, since it has shown slightly better performance than no normalization in our experiments (by contrast, normalizing by ℓ_1 or ℓ_2 norms worsens performance). One-versus-all classification is performed by training one linear SVM for each class using LIBSVM (Chang and Lin, 2001), and then taking the highest score to assign a label to the input. When local pooling in the configuration space is used ($P \geq 1$), clustering is performed using the K-means algorithm to obtain cluster centers. Following the usual practice (Griffin et al., 2007; Lazebnik et al., 2006; Wang et al., 2010), we use

30 training images on the Caltech-101 and Caltech-256 datasets, 100 training images on the Scenes dataset; the remaining images are used for testing, with a maximum of 50 and 20 test images for Caltech-101 and Caltech-256, respectively. Experiments are run ten times on ten random splits of training and testing data, and the reported result is the mean accuracy and standard deviation of these runs. Hyperparameters of the model (such as the regularization parameter of the SVM or the λ parameter of sparse coding) are selected by cross-validation within the training set. Patterns of results are very similar for all three datasets, so results are shown only on Caltech-101 for some of the experiments; more complete numerical results on all three datasets can be found in the Appendix.

5.4.1 Pooling locally in configuration space yields state-of-the-art performance

Experiments presented in Table (5.1) and Table (5.2) compare the performance of sparse coding with a variety of configuration space pooling schemes, with a list of published results of methods using grayscale images and a single type of descriptor. Local pooling always improves results, except on the Scenes for a dictionary of size $K = 1024$. On the Caltech-256 benchmark, our performance of 41.7% accuracy with 30 training examples is similar to the best reported result of 41.2% that we are aware of (for methods using a single type of descriptors over grayscale), obtained by locality-constrained linear codes (Wang et al., 2010), using three scales of SIFT descriptors and a dictionary of size $K = 4096$.

			Caltech 30 tr.	Scenes
$K = 256,$	Pre,	$P = 1$	70.5 ± 0.8	78.8 ± 0.6
		$P = 16$	74.0 ± 1.0	81.5 ± 0.8
		$P = 64$	75.0 ± 0.8	81.1 ± 0.5
		$P = 128$	75.5 ± 0.8	81.0 ± 0.3
		$P = 1 + 16$	74.2 ± 1.1	81.5 ± 0.8
		$P = 1 + 64$	75.6 ± 0.6	81.9 ± 0.7
$K = 256,$	Post,	$P = 16$	75.1 ± 0.8	80.9 ± 0.6
		$P = 64$	76.4 ± 0.8	81.1 ± 0.6
		$P = 128$	$\mathbf{76.7 \pm 0.8}$	81.1 ± 0.5
$K = 1024,$	Pre,	$P = 1$	75.6 ± 0.9	$\mathbf{82.7 \pm 0.7}$
		$P = 16$	76.3 ± 1.1	$\mathbf{82.7 \pm 0.9}$
		$P = 64$	76.2 ± 0.8	81.4 ± 0.7
		$P = 1 + 16$	$\mathbf{76.9 \pm 1.0}$	$\mathbf{83.3 \pm 1.0}$
		$P = 1 + 64$	$\mathbf{77.3 \pm 0.6}$	$\mathbf{83.1 \pm 0.7}$
$K = 1024,$	Post,	$P = 16$	$\mathbf{77.0 \pm 0.8}$	$\mathbf{82.9 \pm 0.6}$
		$P = 64$	$\mathbf{77.1 \pm 0.7}$	$\mathbf{82.4 \pm 0.7}$

Table 5.1: Results on Caltech-101 (30 training samples per class) and 15-scenes, given as a function of whether clustering is performed before (Pre) or after (Post) the encoding, K: dictionary size, and P: number of configuration space bins. Results within one standard deviation of the best results are all shown in bold.

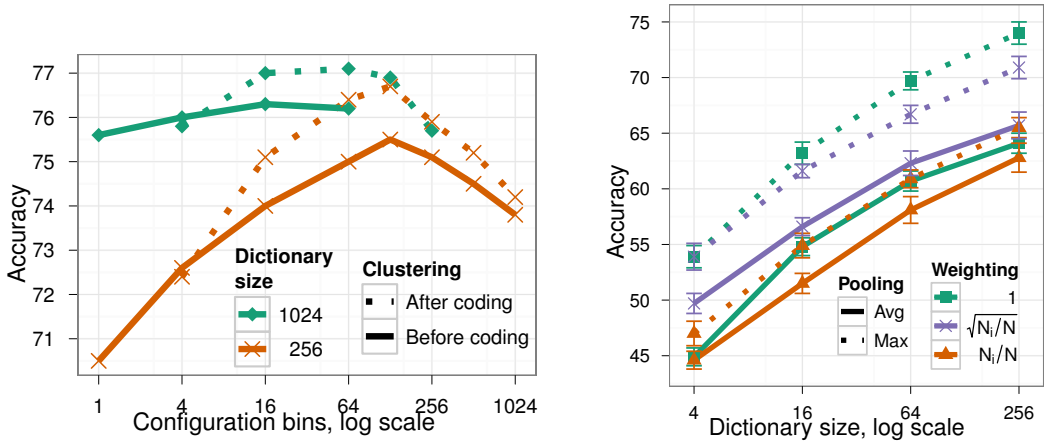
			Accuracy
Boiman et al. (Boiman et al., 2008)			37.0
Gao et al. (Gao et al., 2010) ($K = 1024$)			35.7 ± 0.1
Kim et al. (Kim and Grauman, 2010)			36.3
van Gemert et al. (van Gemert et al., 2010) ($K = 128$)			27.2 ± 0.5
Wang et al. (Wang et al., 2010) ($K = 4096$)			$\mathbf{41.2}$
Yang et al. (Yang et al., 2009b) ($K = 1024$)			34.0 ± 0.4
$K = 256,$	Pre,	$P = 1$	32.3 ± 0.8
		$P = 16$	38.0 ± 0.5
		$P = 64$	39.2 ± 0.5
		$P = 128$	39.7 ± 0.6
$K = 256,$	Post,	$P = 16$	36.9 ± 0.7
		$P = 64$	39.6 ± 0.5
		$P = 128$	40.3 ± 0.6
$K = 1024,$	Pre,	$P = 1$	38.1 ± 0.6
		$P = 16$	$\mathbf{41.6 \pm 0.6}$
		$P = 64$	$\mathbf{41.7 \pm 0.8}$
$K = 1024,$	Post,	$P = 16$	40.4 ± 0.6

Table 5.2: Recognition accuracy on Caltech 256, 30 training examples, for several methods using a single descriptor over grayscale. For our method, results are shown as a function of whether clustering is performed before (Pre) or after (Post) the encoding, K: dictionary size, and P: number of configuration space bins.

Using pyramids in configuration space

We examine whether it is advantageous to combine fine and coarse clustering, in a way reminiscent of the levels of the spatial pyramid. With large dictionaries, local pooling in the configuration space does not always perform better than standard global pooling (see Table (5.1) and Table (5.2)). However, combining levels of different coarseness gives performance better than or similar to that of the best individual level, as has been observed with the spatial pyramid (Lazebnik et al., 2006).

This significantly improves performance on the Caltech-101 dataset. To the best of our knowledge, our performance of 77.3% on the Caltech-101 benchmark, was above all previously published results for a single descriptor type using grayscale images at the time of publication of our paper (Boureau et al., 2011) – although better performance has been reported with color images (e.g., $78.5\% \pm 0.4$ with a saliency-based approach (Kanan and Cottrell, 2010)), multiple descriptor types (e.g., methods using multiple kernel learning have achieved $77.7\% \pm 0.3$ (Gehler and Nowozin, 2009), $78.0\% \pm 0.3$ (VGG Results URL, 2012; Vedaldi et al., 2009) and 84.3% (Yang et al., 2009a) on Caltech-101 with 30 training examples), or subcategory learning (83% on Caltech-101 (Todorovic and Ahuja, 2008)). On the Scenes benchmark, preclustering does improve results for small dictionaries ($K \leq 256$, see Appendix), but not for larger ones ($K = 1024$). While our method outperforms the Laplacian sparse coding approach (Gao et al., 2010) on the Caltech 256 dataset, our performance is much below that of Laplacian sparse coding on the Scenes database.



(a) (b)

Figure 5.2: Recognition accuracy on Caltech-101. Left: Clustering after the encoding generally performs better; for both schemes, binning too finely in configuration space (large P) hurts performance. Right: the best performance is obtained with max pooling and uniform weighting. Max pooling consistently outperforms average pooling for all weighting schemes. With average pooling, weighting by the square root of the cluster weight performs best. $P = 16$ configuration space bins are used. Results on the Caltech-256 and Scenes datasets show similar patterns. Best viewed in color.

Pre- vs. post-clustering

One advantage of using the same dictionary for all features is that the clustering can be performed after the encoding. The instability of sparse coding could cause features similar in descriptor space to be mapped to dissimilar codes, which would then be pooled together. This does not happen if clustering is performed on the codes themselves. While pre-clustering may perform better for few clusters, post-clustering yields better

results when enough clusters are used ($P \geq 64$); a dictionary of size $K = 1024$ reaches 77.1 ± 0.7 accuracy on Caltech-101 with $P = 64$ bins (to be compared to 76.2 ± 0.8 when clustering before coding), while a dictionary of size $K = 256$ yields 76.7 ± 0.8 with $P = 128$ bins (to be compared to 75.5 ± 0.8 with preclustering). Fig. (5.2(a)) also shows that performance drops for larger P, irrespective of whether the clustering is performed before or after the encoding.

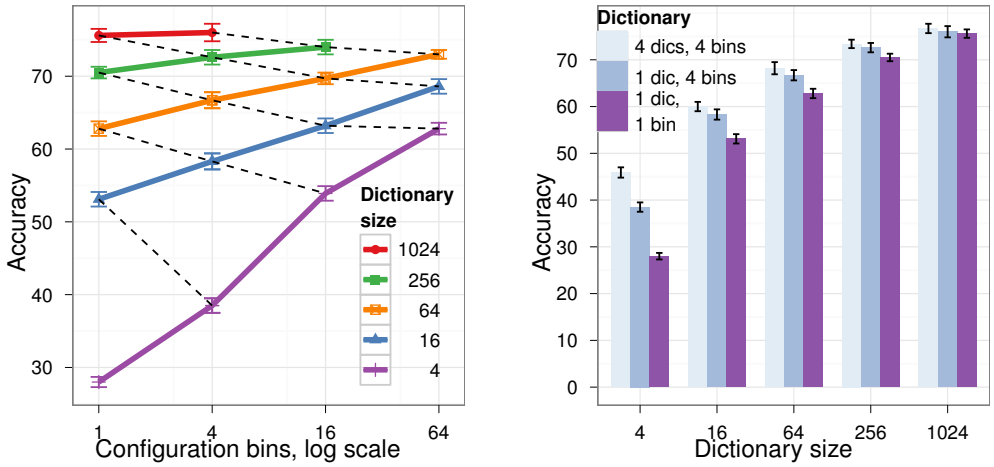
5.4.2 Gaining a finer understanding of local configuration space pooling

In this section, we investigate how much local configuration space pooling can enhance the performance of small dictionaries, how it compares to learning one local dictionary per configuration bins, and what pooling and weighting schemes work best in our pipeline.

Local pooling boosts small dictionaries

Fig. (5.3(a)) shows results for various assignments of components between atoms (K) and centroids (P). Pooling more locally in configuration space ($P > 1$) can considerably boost the performance of small dictionaries.

Unsurprisingly, larger dictionaries consistently beat smaller ones combined with pooling using local configuration bins, at same total number of components; this can be seen from the downwards slope of the gray dashed lines in Fig. (5.3(a)) linking data points at constant $K * P$. However, if P is allowed to grow more, small dictionaries can outperform larger ones. This leads to good performance with a small dictionary; e.g., a



(a) (b)

Figure 5.3: Recognition accuracy on Caltech-101. Left: pooling locally in finer configuration space bins can boost the performance of small dictionaries. Dotted gray lines indicate constant product of dictionary size $\times$ number of configuration bins. Right: a substantial part of the improvement observed when using multiple local dictionaries can be achieved without changing the encoding, by pooling locally in configuration space. $P = 4$ configuration space bins are used. Best viewed in color.

dictionary of just $K = 64$ atoms coupled with a preclustering along $P = 64$ centroids achieves $73.0 \pm 0.6\%$ on Caltech-101.

Comparison with cluster-specific dictionaries

In addition to learning richer, more local dictionaries, learning one dictionary per cluster as done in (Wang et al., 2010; Yang et al., 2010) inherently leads to more local pooling. Experiments in this section seek to disentangle these effects. As shown in Fig. (5.3(b)), more than half of the improvement compared to no preclustering is usually due to the

separate clustering rather than more specific dictionaries. The smaller the dictionary, the larger the proportion of the improvement due to clustering. This may be due to the fact that smaller dictionaries do not have enough atoms to implicitly link activation of an atom to cluster membership during coding, leaving more of that task to the explicit local configuration space pooling than when large dictionaries are used.

Pooling operator and cluster weighting

When concatenating the vectors corresponding to each pool, it is not clear whether they should be weighted according to the prominence of the cluster, measured as the ratio N_i/N of the number N_i of inputs falling into cluster i, over the total number N of inputs. Denoting by w_i the weight for cluster i, we compare three weighting schemes: identical weight ($w_i = 1$), a weight proportional to the square root of the ratio ($w_i = \sqrt{N_i/N}$) as proposed by Zhou et al. (Zhou et al., 2010), or the ratio itself ($w_i = N_i/N$).

As shown in Fig. (5.2(b)), the weighting scheme assigning the same weight to each cluster performs better when max pooling is used, except for very small dictionaries. When average pooling is used, the best weighting scheme is the square root weighting, which empirically validates the choice in (Zhou et al., 2010), but performance is below that of max pooling. Based on these results, max pooling with identical weighting for all clusters has been used for all other experiments in the chapter.

5.5 Conclusion

While there is no question that making coding more stable and more specific is advantageous, the simple procedure of clustering the data in order to make pooling local in

configuration space is a powerful tool for image recognition. The main conclusions of this work are that (1) more local configuration space pooling in itself boosts performance, dramatically so with smaller dictionaries; (2) it is advantageous to use pyramids rather than grids, analogously to spatial pooling; (3) with enough configuration space bins, better performance may be obtained when the clustering is performed just before the pooling step, rather than before the coding step; (4) performance drops if too many bins are added.

6

MAKING CODING REAL-TIME AND CONVOLUTIONAL

As seen in the previous chapters, ℓ_1-regularized sparse coding is a powerful method for image applications. However, its computational cost remains too high for real-time applications. This chapter presents several viable alternatives that have offered significant speed-up without leading to dramatic loss of performance. Another shortcoming of sparse coding is that dictionaries are trained over isolated patches, but coding is then performed like a convolution (i.e., as a sliding window). We present convolutional sparse coding training methods that are more suited to the redundancy of a sliding window setting. Some of the research presented in this chapter has been published in (Ranzato et al., 2007a; Ranzato et al., 2007c; Ranzato et al., 2007b; Kavukcuoglu et al., 2010).

6.1 Introduction

The architectures explored in this thesis that obtain the best performance rely on solving an ℓ_1-regularized optimization. Several efficient algorithms have been devised for this problem. Homotopy methods such as the LARS algorithm (Efron et al., 2004) give a set of solutions along the regularization path (i.e., for a range of sparsity penalties), and can be very fast when implemented well, if the solution is sufficently sparse. Coordinate descent is fast in practice, as recently rediscovered (Friedman et al., 2007; Li and

Osher, 2009; Friedman et al., 2010). Accelerated gradient methods (Beck and Teboulle, 2009) offer additional speed-ups. However sparse coding is still too slow for real-time applications.

One strategy is to get rid of the ℓ_1 regularization and obtain sparse codes in a different way, for example with greedy solutions to the ℓ_0 sparse coding problem such as orthogonal matching pursuit (OMP) (Mallat and Zhang, 1993), or by using a simple thresholding criterion over the dot-products of the input with the dictionary elements. Experiments with these techniques are shown in Sec. (6.2). Another option is to turn to methods that have been known to work well in real-time settings, namely, neural network architectures, and find some way to encourage them to produce sparse activations. In Sec. (6.3), we incorporate one layer of a feedforward encoder to replace the sparse coding module in the spatial pyramid framework.

Another problem of many unsupervised training modules is that training and inference are performed at the patch level. In most applications of sparse coding to image analysis (Aharon et al., 2005; Mairal et al., 2009a), the system is trained on *single image patches* whose dimensions match those of the filters. Inference is performed on all (overlapping) patches independently, which produces a highly redundant representation for the whole image, ignoring the fact that the filters are 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 be able to reconstruct each patch in isolation. Note that this is not a problem for SIFT descriptors, since the detection of orientation is designed to be centered in SIFT. We present convolutional versions of unsupervised training algorithms in Sec. (6.3.1) for convolutional RBMs, and Sec. (6.4) for sparse feedforward encoders.

6.2 Sparse coding modules without ℓ_1

ℓ_1 and ℓ_0 regularizations have different but complementary strengths and weaknesses. ℓ_0 produces sparse solutions with an obvious relationship between the regularization coefficient and the sparsity of the solution, and greedy ℓ_0 approximations such as OMP (Mallat and Zhang, 1993) perform well. On the downside, the codes produced are very unstable, and not as good for training a dictionary. ℓ_1 regularization produces good dictionaries, but induces a shrinkage of the solution. Combining both ℓ_0 and ℓ_1 can give the best of both worlds; denoising methods based on sparse coding thus obtain their best results when training a dictionary with an ℓ_1 penalty, then using it to reconstruct the patches with an ℓ_0 penalty (Mairal et al., 2009b). We do the same here, and report results using ℓ_0 greedy optimization over a dictionary trained with an ℓ_1 penalty.

A cruder way to obtain sparse coefficients is to simply compute dot-products of the input with the dictionary atoms, and apply a threshold. The resulting code cannot be used to produce good reconstructions, but the sparse representation may be used for other tasks such as classification.

We have compared these alternatives to ℓ_1-regularized inference, on the Caltech-101 and Scenes datasets. Results are presented in Table (6.1), and include experiments with feedforward encoders discussed in the next section for ease of comparison. Both greedy inference using an ℓ_0 penalty and thresholded dot-products lose some accuracy compared to inference with the ℓ_1 penalty, but they are much faster.

Recent work on other datasets has shown that simple thresholding schemes can outperform sparse coding in some cases (Coates et al., 2011; Coates and Ng, 2011).

	15 tr.	30 tr.
ℓ_1-regularized, $K = 512$	63.5 ± 0.8	70.3 ± 1.0
RBM, $K = 768$	51.8 ± 1.2	59.3 ± 0.9
K=1024		
ℓ_1-regularized	64.7 ± 1.0	71.5 ± 1.1
Trained encoder	62.5 ± 1.4	69.6 ± 1.0
Thresholded dot-product with sparse dictionary	62.1 ± 1.5	69.5 ± 1.0
ℓ_0-regularized	62.3 ± 0.9	69.9 ± 1.0

Table 6.1: Recognition accuracy on the Caltech-101 dataset, for ℓ_1-regularized sparse coding and several fast alternatives: OMP (ℓ_0-regularized), thresholded dot-products, feedforward encoders trained to reconstruct sparse codes, restricted Boltzmann machines (RBMs). All results are obtained with standard features extracted every 8 pixels, with a 4×4 pyramid, max pooling, linear kernel, using 15 or 30 training images per class, K : dictionary size.

6.3 Single-layer feedforward unsupervised feature extraction modules

We present several trainable feedforward non-linear encoder modules that can produce a fast approximation of the sparse code.

$K = 1024$, linear kernel	
ℓ_1-regularized	83.1 ± 0.6
ℓ_0-regularized	82.3 ± 0.5
Thresholded dot-product with sparse dictionary	80.6 ± 0.4
Trained encoder and decoder	77.6 ± 0.7
RBM	70.9 ± 0.8
$K = 1024$, intersection kernel	
ℓ_1-regularized	84.1 ± 0.5
ℓ_0-regularized	82.8 ± 0.6
RBM	76.3 ± 0.7
$K = 2048$, linear kernel	
ℓ_1-regularized	83.6 ± 0.5
ℓ_0-regularized	83.2 ± 0.5
$K = 4096$, linear kernel	
ℓ_1-regularized	83.8 ± 0.5
thresholded dot-product with sparse dictionary	81.6 ± 0.3

Table 6.2: Recognition accuracy on the Scenes dataset, for ℓ_1-regularized sparse coding and several fast alternatives: OMP (ℓ_0-regularized), thresholded dot-products, feed-forward encoders trained to reconstruct sparse codes, restricted Boltzmann machines (RBMs). All results are obtained with standard features extracted every 8 pixels, with a 4×4 pyramid, max pooling using 100 training images per class,

6.3.1 Restricted Boltzmann machines (RBMs)

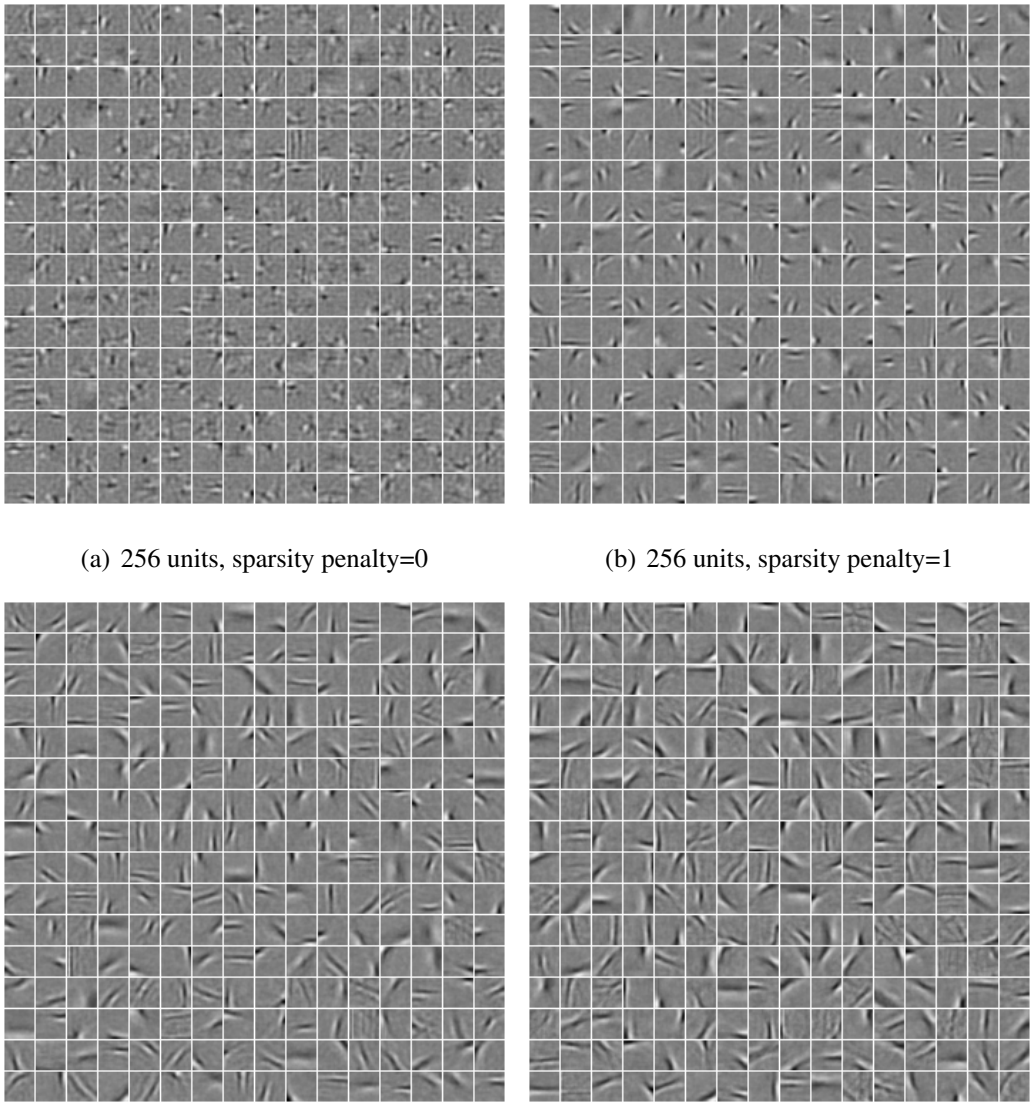
RBMs have been introduced in Sec. (1.3).

Adding a sparsity penalty to RBMs. Lee et al. have proposed adding a sparsity penalty to the RBM loss function, and have obtained second-layer units sharing many properties with neurons of the V2 area (Lee et al., 2007).

To illustrate the effect of that sparsity penalty, we have trained filters on 21×21 patches of the Caltech-101 dataset, varying both the number of units and the weight of the sparsity penalty. Learned filters are shown in Fig. (6.1) and Fig. (6.2). The edges get longer when the sparsity term is weighted more; looking at filters during training, we also observed that the edges emerged faster. The smaller the number of hidden units, the higher the sparsity penalty has to be pushed to produce edges. Training a sufficiently overcomplete set of units in an RBM produces stroke detectors even without a sparsity penalty. A more extensive set of filter images can be found in the Appendix.

Using RBMs to replace the sparse coding module. Using the SIFT descriptor maps as inputs, we have used sparse RBMs to replace the sparse coding modules in our experiments. Results are shown in Table (6.1). In these experiments, the performance of RBMs is not competitive with other approaches.

Convolutional extension. Many of RBM filters trained on patches are merely translated verisons of each other (Fig. (6.1) and Fig. (6.2)). This is wasteful for the same arguments as with patch-based sparse coding if the RBM filters are then used in a sliding-window setting. In a dataset with a strong bias for vertical and horizontal image struc-



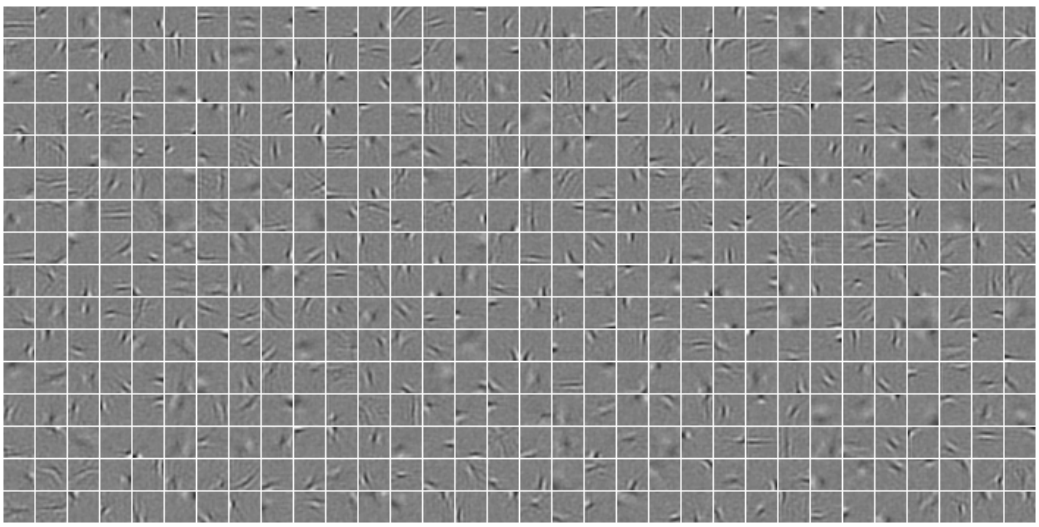
(a) 256 units, sparsity penalty=0

(b) 256 units, sparsity penalty=1

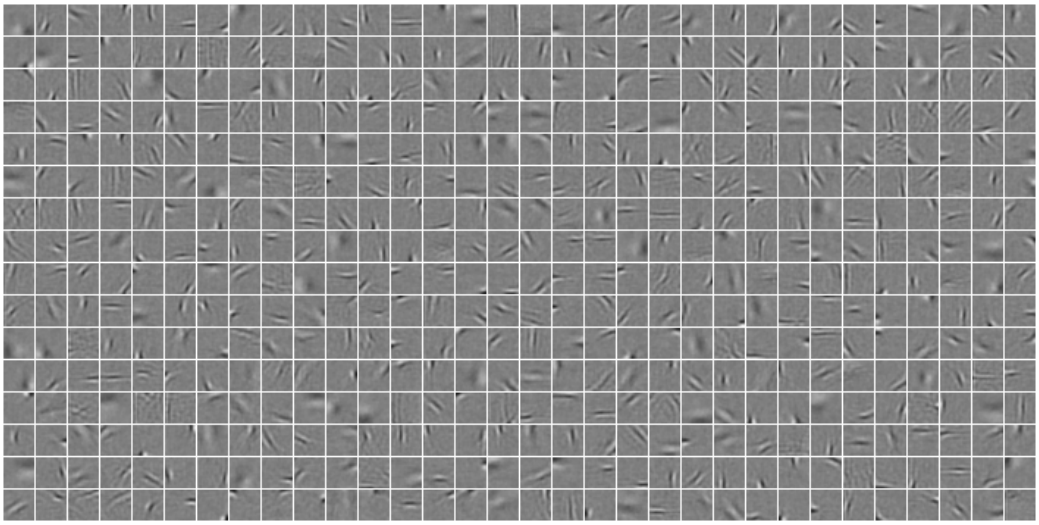
(c) 256 units, sparsity penalty=5

(d) 256 units, sparsity penalty=25

Figure 6.1: 256 RBM hidden units trained with increasingly weighted sparsity penalty over 21×21 patches of the Caltech-101 dataset.



(a) 512 units, sparsity penalty=0



(b) 512 units, sparsity penalty=1

Figure 6.2: 512 RBM hidden units trained with increasingly weighted sparsity penalty over 21×21 patches of the Caltech-101 dataset.

tures, the machine will even learn a collection of vertical bars at all positions rather than adding an oblique filter, since training does not incorporate all the information that will be available from neighboring patches during utilisation.

We briefly present here some results using convolutional training for RBMs. Since that work, research showing promising applications of convolutional RBMs for recognition has been published (Lee et al., 2009). Convolutional training can be conducted in a way similar as with sparse autoencoders, by using larger patches during training. Denote by P the length of the larger patch used to explain a central patch of size $p \times p$. We take $P = 3p - 2$, so as to cover all hidden units that would be connected to any visible unit of the central patch of interest.

The convolutional training algorithm is then the same as the standard RBM training one with the following alterations:

1. The visible and hidden units are connected as they would be when running a patch-based RBM in a sliding window

2. Hidden unit activations are computed conditioned on the visible activations of the larger patch of length P

3. When computing the products $v_i h_j$ for a pair of visible unit i and hidden unit j to update the weights, only take into account the visible units of the central smaller patch v_i.

We train these convolutional RBMs on stills from a video clip. The training data consists of patches of frames from a National Geographic video downloaded from YouTube. The 3200 360×480 frames of the movie are spatially downsampled to 72×96 pixels.

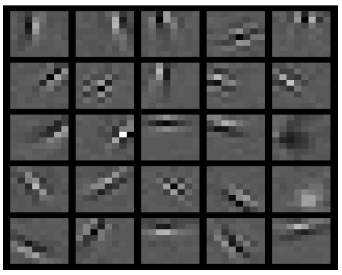


Figure 6.3: 25 8x8 filters extracted from stills from a video clip. They all have either different orientation or phase

Random patches are extracted and whitened to obtain patches of size 22×22. We train 25 8×8 RBM filter from these patches. Learned filters are shown in Fig. (6.3). A varied set of orientations are obtained.

We have also trained convolutional RBMs over spatio-temporal patches from videos to obtain 25 hidden units connected to $8 \times 8 \times 5$ visible units. An example application of these units is to upsample the frames of a movie, as shown in Fig. (6.4), where a YouTube video of a cat in a tub has been upsampled using the spatiotemporal units trained on the unrelated National Geographic video clip with higher sampling rate. Each 5-frame sequence is obtained from an input of frames 1 and 5.

6.3.2 Sparse autoencoders

Feedforward sparse coding by approximating sparse codes

Feedforward architectures used to predict sparse codes (Kavukcuoglu et al., 2008; Ranzato et al., 2007b; Jarrett et al., 2009) usually infer a sparse code during training that needs to be close to the code predicted by the encoder; hence it is not the optimal re-

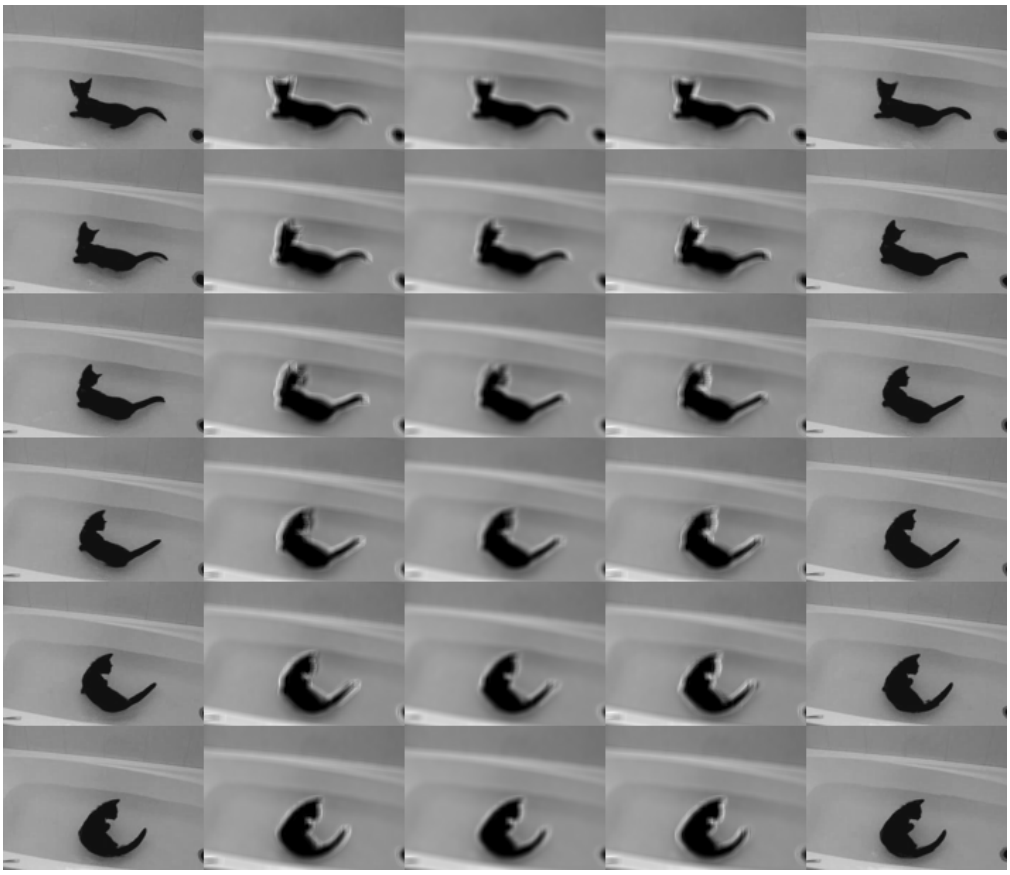


Figure 6.4: Upsampling a video. Frames 1 and 5 of each sequence are input; the machine bridges the gap between them.

Coding	15 tr.	30 tr.
Hard vector quantization	55.9	63.8
Soft vector quantization	57.9	65.6
Sparse coding		
on k-mean centers	60.7	66.2
on learned dictionary	61.7	68.1
feedforward encoder	60.5	68.0

Table 6.3: Recognition accuracy on the Caltech-101 dataset, 4x4 grid, dictionary of size $K = 200$, average pooling, intersection kernel, using 15 or 30 training images per class.

constructive sparse code. We adopt the same idea of learning a predictor, but train it directly to predict the sparse codes obtained from the dictionaries trained for a purely ℓ_1-regularized reconstruction loss.

Denoting the representation of input $\boldsymbol{x}_i$ by $\boldsymbol{z}_i$, and the logistic function by $\sigma(t) = \frac{1}{1+e^{-t}}$, a weight matrix $\mathbf{W} \in \mathbf{R}^{K \times M}$ and a gain vector $\mathbf{g} \in \mathbf{R}^K$ are obtained by minimizing the squared difference with inferred sparse codes:

$$(\mathbf{D}^*, \boldsymbol{\alpha}_1^*, \ldots, \boldsymbol{\alpha}_N^*) = \underset{\mathbf{D}, \boldsymbol{\alpha}_1, \ldots, \alpha_N}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{x}_i - \mathbf{D}\boldsymbol{\alpha}_i\|_2^2 + \lambda \|\boldsymbol{\alpha}_i\|_1, \tag{6.1}$$

$$(\mathbf{W}^*, \mathbf{g}^*) = \underset{\mathbf{W}, \mathbf{g}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{diag}(\mathbf{g})\sigma(\mathbf{W}\boldsymbol{x}_i) - \boldsymbol{\alpha}_i^*\|_2^2, \tag{6.2}$$

$$\boldsymbol{z}_i = \mathbf{diag}(\mathbf{g}^*)\sigma(\mathbf{W}^*\boldsymbol{x}_i). \tag{6.3}$$

Using this encoder, results can be as good as with the inferred sparse code when the dictionary is small, as shown in Table (6.3). However, performance is again below

sparse codes for larger dictionaries, and even below the simple thresholded dot-product with the sparse dictionary on the Scenes dataset (see Table (6.1) and Table (6.2)).

6.4 Making training convolutional

As argued in the introduction of this chapter, patch-based training does not take into account the redundancy inherent to convolutional schemes. To make sparse coding convolutional, we apply it to the entire image at once, and view the dictionary as a convolutional filter bank:

$$\mathcal{L}(x, z, \mathcal{D}) = \frac{1}{2}||x - \sum_{k=1}^{K} \mathcal{D}_k * z_k||_2^2 + |z|_1, \tag{6.4}$$

where $\mathcal{D}_k$ is an $s \times s$ 2D filter kernel, x is a $w \times h$ image (instead of an $s \times s$ patch), z_k 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 proposed before (Zeiler et al., 2010), but in a setting that does not allow real-time processing. To make convolutional sparse coding faster, train a feedforward, non-linear *encoder* module to produce a fast approximation of the sparse code, similar to (Ranzato et al., 2007b; Kavukcuoglu et al., 2009). The new energy function includes a code prediction error term:

$$\mathcal{L}(x, z, \mathcal{D}, W) = \frac{1}{2}||x - \sum_{k=1}^{K} \mathcal{D}_k * z_k||_2^2 + \sum_{k=1}^{K} ||z_k - f(W^k * x)||_2^2 + |z|_1, \tag{6.5}$$

where $z^* = \text{argmin}_z \mathcal{L}(x, z, \mathcal{D}, W)$ and W^k is an encoding convolution kernel of size $s \times s$, and f is a point-wise non-linear function. Two important questions are the form of the non-linear function f, and the optimization method to find z^*.

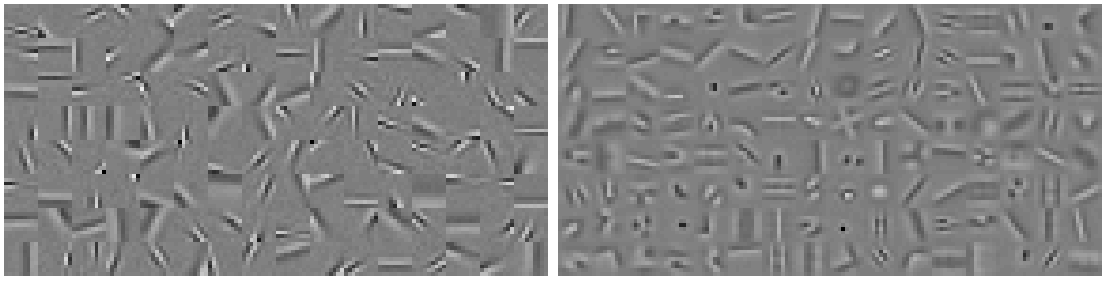


Figure 6.5: **Left:** A dictionary with 128 elements, learned with patch based sparse coding model. **Right:** A dictionary with 128 elements, learned with convolutional sparse coding model. The dictionary learned with the convolutional model spans the orientation space much more uniformly. In addition it can be seen that the diversity of filters obtained by convolutional sparse model is much richer compared to patch based one.

6.5 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, 2009) algorithm and the dictionary learning procedure.

6.5.1 Learning convolutional dictionaries

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. Decomposition can then be done with a standard sparse coding algorithm. Unfortunately, the size of the dictionary then depends on the size of the input signal. This argues for a formulation based on

convolutions. In this work, we use the coordinate descent sparse coding algorithm (Li and Osher, 2009) 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 Eq. (6.4) 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., 2009a; Ranzato et al., 2007b)). One can use either batch (Aharon et al., 2005) (by accumulating derivatives over many samples) or online updates (Mairal et al., 2009a; 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 Eq. (6.4), 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 with respect to 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.

The second important point in training convolutional dictionaries is the computa-

Algorithm 1 Convolutional extension to coordinate descent sparse coding(Li and Osher, 2009). 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 $\mathbf{ConvCoD}(x, \mathcal{D}, \alpha)$

Set: $S = \mathcal{D}^T * \mathcal{D}$

Initialize: $z = 0$; $\beta = \mathcal{D}^T * mask(x)$

Require: h_α : smooth thresholding function.

repeat

$\bar{z} = h_\alpha(\beta)$

$(k, p, q) = \text{argmax}_{i,m,n} |z_{imn} - \bar{z}_{imn}|$ (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

tion of the $S = D^T * D$ operator. In algorithms such as coordinate descent (Li and Osher, 2009), accelerated proximal methods (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 used with some additional care: instead of being the dot-product of dictionary atoms i and j, each term has to be expanded as *"full"* convolution of two dictionary elements (i, j), producing a $(2s - 1) \times (2s - 1)$ matrix. One can think about the resulting matrix as a $4D$ tensor of

size $K \times K \times (2s-1) \times (2s-1)$. The overall learning procedure is simple stochastic (online) gradient descent over dictionary $\mathcal{D}$:

$$\forall x^i \in \mathcal{X} \text{ training set}: \quad z^* = \underset{z}{\operatorname{argmin}}\, \mathcal{L}(x^i, z, \mathcal{D}), \quad \mathcal{D} \leftarrow \mathcal{D} - \eta \frac{\partial \mathcal{L}(x^i, z^*, \mathcal{D})}{\partial \mathcal{D}} \quad (6.6)$$

The columns of $\mathcal{D}$ are normalized after each iteration. A convolutional dictionary with 128 elements trained on images from Berkeley dataset (Martin et al., 2001) is shown in Fig. (6.5).

6.5.2 Learning an efficient encoder

In (Ranzato et al., 2007b), (Jarrett et al., 2009) and (Gregor and LeCun, 2010) a feed-forward 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: $\tilde{z} = g^k \times tanh(x * W^k) \;\; (k = 1..K)$. 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. 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 $\tilde{z} = sh_{\beta^k, b^k}(x * W^k)$ where $k = 1..K$ and

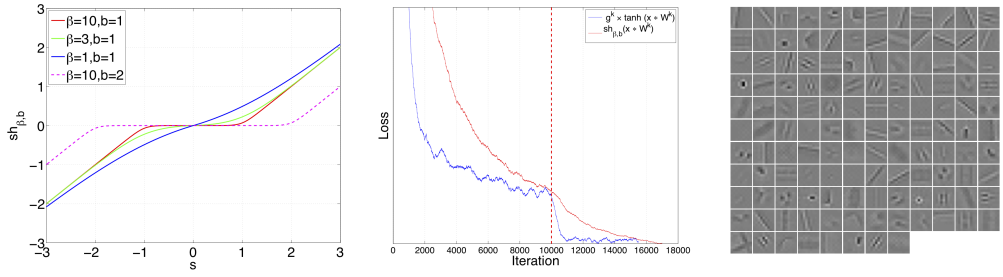


Figure 6.6: **Left:** Smooth shrinkage function. Parameters β and b control the smoothness and location of the kink of the function. As $\beta \to \infty$ it converges more closely to soft thresholding operator. **Center:** Total loss as a function of number of iterations. The vertical dotted line marks the iteration number when diagonal hessian approximation was updated. It is clear that for both encoder functions, hessian update improves the convergence significantly. **Right:** 128 convolutional filters (W) learned in the encoder using smooth shrinkage function. The decoder of this system is shown in Fig. (6.5).

:

$$sh_{\beta^k,b^k}(s) = sign(s) \times 1/\beta^k \log(\exp(\beta^k \times b^k) + \exp(\beta^k \times |s|) - 1) - b^k \qquad (6.7)$$

Note that each β^k and b^k is a singleton per each feature map k. The shape of the smooth shrinkage operator is given in Fig. (6.6) for several different values of β and b. β 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 Eq. (6.5).

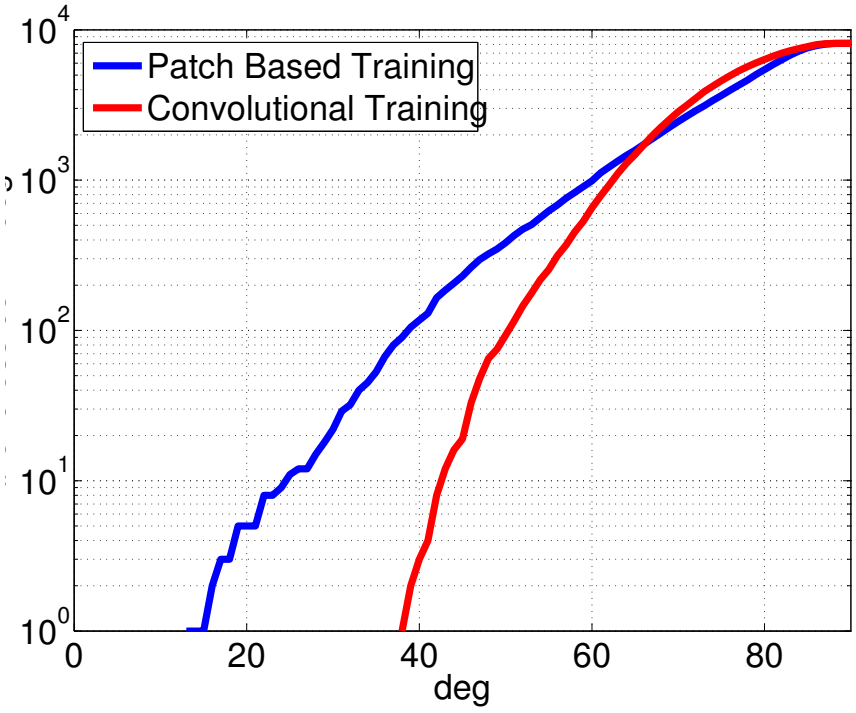


Figure 6.7: Cumulative histogram of angles between dictionary item pairs. The minimum angle with convolutional training is 40 degrees.

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 Fig. (6.6). It can be seen that especially for the $tanh$ encoder function, the effect of using second order information on the convergence is

significant.

6.5.3 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 $k \times k$ 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 the oriented structure once, since dictionary elements can be used at all locations. Fig. (6.5) 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. Dictionary atoms are more dissimilar, as measured by their angle (Fig. (6.7)).

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

6.5.4 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 (Jarrett et al., 2009). Building on previous findings in that paper, for each stage, the encoder is followed by 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 9×9 neighborhood in spatial dimensions. The last operation, **average pooling** is simply a spatial pooling operation that is applied on each feature map independently. A complete stage is illustrated in Fig. (6.8).

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.

6.6 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.

6.6.1 Object recognition using the Caltech-101 dataset

We again use the Caltech-101 dataset (Fei-Fei et al., 2004). 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 9×9 neighborhood. **3.** The short side of each image is zero padded to 143 pixels. We report the results in Table (6.4) and Table (6.5). All results in these tables are obtained using 30 training images per class and 5 different splits of the data. 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 segmen-

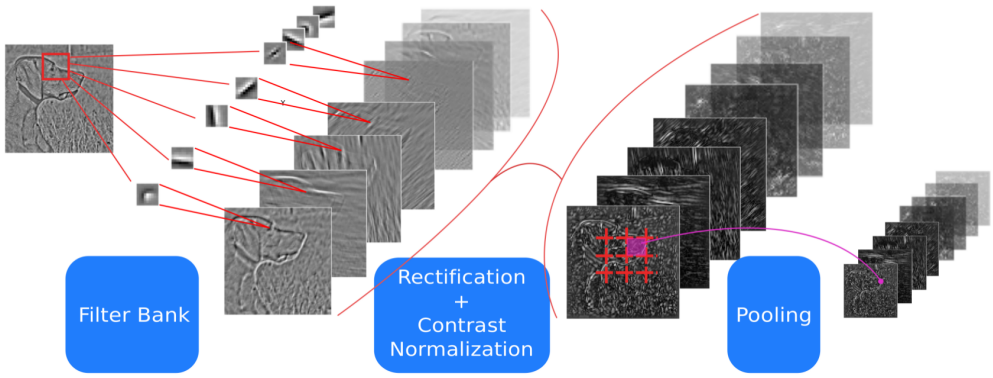


Figure 6.8: Architecture of one stage of feature extraction.

tation 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 9×9 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 9×9 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.

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%

Table 6.4: 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.

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 average pooling. The convolutional filters used are 9×9. The average pooling is applied over a 10×10 area with 5 pixel stride. The output of first layer is then $64 \times 26 \times 26$ and fed into a logistic regression classifier, or used as lower level of a standard spatial pyramid pipeline (hard vector quantization, pooling over a 4×4 pyramid (Lazebnik et al., 2006)), instead of the SIFT descriptor layer.

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 9×9, producing $256 \times 18 \times 18$ features. After applying absolute value rectification, contrast normalization and average pooling (on a 6×6 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.

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

Table 6.5: Recognition accuracy on Caltech 101 dataset using a variety of different feature representations using two stage systems and two different classifiers.

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%). We get 65.7% with a spatial pyramid on top of our single-layer U system (with 256 codewords jointly encoding 2×2

neighborhoods of our features by hard quantization, then max pooling in each cell of the pyramid, with a linear SVM, as proposed in Chapter 3.

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 (6.5).

7

SUPERVISED TRAINING

All methods used in experiments so far have been unsupervised, with supervised information being available only for training the classifier layer: adaptive parameters are trained by minimizing a regularized reconstruction error, without any information about a label. While this ensures that the weights are adapted to the statistics of the data, they are not optimized for the classification task. This chapter incorporates supervision into the architectures discussed and presents experimental results showing the superiority of discriminatively trained parameters. The work presented here has been published in (Boureau et al., 2010a; Kavukcuoglu et al., 2010).

7.1 Supervised training in deep networks

Early deep neural networks for recognition (LeCun et al., 1998a) were trained in a purely supervised manner, by backpropagating the gradient of the unsupervised loss through all layers of the network. The recent innovation has been to use unsupervised training to *initialize* the weights, so that supervised training does not get trapped in local minima as with random initialization (Tesauro, 1992). But the final training is usually supervised.

This section compares the performance of unsupervised weights (i.e., supervised training is confined to the top classifier layer), to performance of "fine-tuned" weights (i.e., the supervised gradient is propagated down lower layers as well), using the convolutional sparse coding architectures presented in Sec. (6.4). The experimental settings

are identical. We compare single and two-stage architectures of convolutional sparse coding with non-convolutional versions, with a logistic regression classifier or a spatial pyramid pipeline (using the trained features to replace the SIFT descriptor layer).

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 7.1: 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.

Results are presented in Table (7.1) and Table (7.2). We denote unsupervised trained one stage systems with U, two stage unsupervised trained systems with UU and "$^{+}$" signals that supervised training has been performed. Several points can be made:

- Supervised finetuning usually improves performance; but two layers of purely unsupervised features (UU, 65.3%) outperform the patch-based model of Jarrett et al. (Jarrett et al., 2009) after supervised fine-tuning (63.7%), even though the models are extremely similar. This shows that finding a better region of parameter space thanks to well-designed unsupervised training may be more important than minimizing the true loss of interest.

- Supervised fine-tuning ($U^{+}U^{+}$) allows our network to match or perform very close to (66.3%) similar models (Lee et al., 2009; Zeiler et al., 2010) with two layers

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

Table 7.2: Recognition accuracy on Caltech 101 dataset using a variety of different feature representations using two stage systems and two different classifiers.

of convolutional feature extraction, even though these models use the a spatial pyramid pipeline (denoted PMK-SVM here) instead of the more basic 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.

- The improvement of convolutional vs. patch-based training is larger when using feature extractors trained in a purely unsupervised way, than when unsupervised training is followed by a supervised training phase. Recalling that the supervised tuning is a *convolutional* procedure, this last training step might have the addi-

tional 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.

7.2 Discriminative dictionaries for sparse coding

We now leave feedforward architectures to go back to variants of bag-of-feature methods.

7.2.1 Previous work

Lazebnik and Raginsky (Lazebnik and Raginsky, 2008) incorporate discriminative information by minimizing the loss of mutual information between features and labels during the quantization step. The method is demonstrated on toy datasets and then used to replace the codebook quantization scheme in a bag-of-features image classifier. This requires computing the conditional distribution of labels and marginal distribution of features. While this is tractable in toy examples where X is low-dimensional, this is clearly not the case for image applications, where the distribution has to be conditioned *on the joint feature space*, which is very high-dimensional. Therefore, in real image applications in (Lazebnik and Raginsky, 2008), the information loss minimization is conducted with distributions conditioned over patches instead of images, each patch being assigned the label of the whole image from which it has been extracted. However, individual patches may contain no discriminative information. For example, in a binary classification case with classes a and b, where the input space consists of two patches

with a single binary feature, with all instances of class a being either $(0,0)$ or $(1,1)$ with equal probability, while all instances of class b are $(0,1)$, the conditional distribution $P(X|Y=a)$ and $P(X|Y=b)$ are the same (with X denoting the single binary feature in any patch). Thus, the individual binary features contain no discriminative information. Conversely, the 2-dimensional bag-of-features representation (X_1, X_2) is discriminative because conditional distributions $P\left((X_1, X_2)|Y=a\right)$ and $P\left((X_1, X_2)|Y=b\right)$ are different.

Similarly, other discriminative approaches ((Mairal et al., 2009c), (Bach and Harchaoui, 2008)) require local patch labels. Winn et al. (Winn et al., 2005) avoid this problem by maximizing the probability of assigning the correct label to the bag of words instead of the individual words. Furthermore, they also solve the potential problem of the lack of spatial correspondance between regions and meaningful codewords by obtaining the regions by hand segmentation, or defining them a posteriori as the ones that show the most clear-cut classification performance. They start with a large dictionary of several thousands atoms obtained by k-means, and prune it iteratively by fusing two codewords (i.e., summing their contributions into the same histogram bin) if they do not contribute to discrimination. But quantization still requires distances to all the original codewords to be computed. We propose to learn directly a discriminative codebook instead of tweaking a huge codebook a posteriori, that is adapted to global image statistics instead of being trained on labelled patches.

7.2.2 Supervised sparse coding

With the same notation as in Chapter 3, let us consider the extraction of a global image representation by sparse coding and average pooling over the whole image I:

$$\hat{\boldsymbol{x}}_i^T = [\boldsymbol{x}_{i_1}^T \cdots \boldsymbol{x}_{i_L}^T], \;\; i_1, \cdots, i_L \in \mathcal{L}_i, \tag{7.1}$$

$$\boldsymbol{\alpha}_i = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \, L(\boldsymbol{\alpha}, \mathbf{D}) \triangleq \frac{1}{2}\|\hat{\boldsymbol{x}}_i - \mathbf{D}\boldsymbol{\alpha}\|_2^2 + \lambda\|\boldsymbol{\alpha}\|_1, \tag{7.2}$$

$$\boldsymbol{h} = \frac{1}{|I|}\sum_{i \in I} \boldsymbol{\alpha_i}, \tag{7.3}$$

$$\boldsymbol{z} = \boldsymbol{h}. \tag{7.4}$$

Consider a binary classification problem. Let $\boldsymbol{z}^{(n)}$ denote the global image representation for the n-th training image, and $y_n \in \{-1, 1\}$ the image label. A linear classifier is trained by minimizing with respect to parameter θ the regularized logistic cost:

$$C_s = \frac{1}{N}\sum_{n=1}^{N} \log\left(1 + e^{-y_n \theta^T \boldsymbol{z}^{(n)}}\right) + \lambda_r \|\theta\|_2^2, \tag{7.5}$$

where λ_r denotes a regularization parameter. We use logistic regression because its level of performance is typically similar to that of linear SVMs but unlike SVMs, its loss function is differentiable. We want to minimize the supervised cost C_s with respect to $\mathbf{D}$ to obtain a more discriminative dictionary. Using the chain rule, we obtain:

$$\frac{\partial C_s}{\partial \mathbf{D}_{jk}} = -\frac{1}{N}\sum_{n=1}^{N} y_n \left(1 - \sigma(y_n \theta . \boldsymbol{z}^{(n)})\right) \theta^T \frac{\partial \boldsymbol{z}^{(n)}}{\partial \mathbf{D}_{jk}} \tag{7.6}$$

$$\frac{\partial \boldsymbol{z}^{(n)}}{\partial \mathbf{D}_{jk}} = \frac{1}{|I^{(n)}|}\sum_{i \in I^{(n)}} \frac{\partial \boldsymbol{\alpha}_i^{(n)}}{\partial \mathbf{D}_{jk}}, \tag{7.7}$$

where σ denotes the sigmoid function $\sigma(x) = 1/(1 + \exp(-x))$. We need to compute the gradient $\nabla_{\mathbf{D}}(\boldsymbol{\alpha}_i)$. Since the $\boldsymbol{\alpha}_i$ minimize Eq. (7.2)), they verify:

$$\boldsymbol{\alpha} = ({\mathbf{D}_{\boldsymbol{\alpha}}}^T \mathbf{D}_{\boldsymbol{\alpha}})^{-1}({\mathbf{D}_{\boldsymbol{\alpha}}}^T \hat{\boldsymbol{x}} - \lambda \text{sign}(\boldsymbol{\alpha})), \tag{7.8}$$

where we have dropped subscript i to limit notation clutter, and $\mathbf{D}_{\boldsymbol{\alpha}}$ denotes the columns corresponding to the active set of $\boldsymbol{\alpha}$ (i.e., the few columns of $\mathbf{D}$ used in the decomposition of the input). Note that this formula cannot be used to compute $\boldsymbol{\alpha}$, since parts of the right-hand side of the equation depend on $\boldsymbol{\alpha}$ itself, but it can be used to compute a gradient once $\boldsymbol{\alpha}$ is known. When perturbations of the dictionary are small, the active set of $\boldsymbol{\alpha}$ often stays the same (since the correlation between the atoms of the dictionary and the input vector varies continuously with the dictionary). Assuming that it is constant, we can compute the gradient of the active coefficients with respect to the active columns of $\mathbf{D}$ (setting it to 0 elsewhere):

$$\frac{\partial \tilde{\boldsymbol{\alpha}}_k}{\partial (\mathbf{D}_{\boldsymbol{\alpha}})_{ij}} = \mathbf{b}_i \mathbf{A}_{kj} - \tilde{\boldsymbol{\alpha}}_j \mathbf{C}_{ki}, \tag{7.9}$$

$$\mathbf{A} \triangleq ({\mathbf{D}_{\boldsymbol{\alpha}}}^T \mathbf{D}_{\boldsymbol{\alpha}})^{-1}, \tag{7.10}$$

$$\mathbf{b} \triangleq \hat{\boldsymbol{x}} - \mathbf{D}\boldsymbol{\alpha}, \tag{7.11}$$

$$\mathbf{C} \triangleq \mathbf{A}{\mathbf{D}_{\boldsymbol{\alpha}}}^T, \tag{7.12}$$

where $\tilde{\boldsymbol{\alpha}}_k$ denotes the k-th non-zero component of $\boldsymbol{\alpha}$.

We train the discriminative dictionary by stochastic gradient descent (Bottou, 1998; LeCun et al., 1998b). Recomputing the sparse decompositions $\boldsymbol{\alpha}_i$ at each location of a training image at each iteration is costly. To speed-up the computation while remaining closer to global image statistics than with individual patches, we approximate $\boldsymbol{z}^{(n)}$ by

	Unsup	Discr	Unsup	Discr
Linear	83.6 ± 0.4	$\mathbf{84.9 \pm 0.3}$	84.2 ± 0.3	$\mathbf{85.6 \pm 0.2}$
Intersect	84.3 ± 0.5	$\mathbf{84.7 \pm 0.4}$	84.6 ± 0.4	85.1 ± 0.5

Table 7.3: Results of learning discriminative dictionaries on the Scenes dataset, for dictionaries of size 1024 (left) and 2048 (right), with 2×2 macrofeatures and grid resolution of 8 pixels,

pooling over a random sample of ten locations of the image. Furthermore, we update only a random subset of coordinates at each iteration, since computation of the gradient is costly. We then test the dictionary with max pooling and a three-layer spatial pyramid, using either a linear or intersection kernel SVM.

We compare performance of dictionaries of sizes 1024 and 2048 on the Scenes dataset, encoding 2×2 neighborhoods of SIFT. Results (Table (7.3)) show that discriminative dictionaries perform significantly better than unsupervised dictionaries. A discriminative dictionary of 2048 codewords achieves 85.6% correct recognition performance, which to the best of our knowledge was the highest published classification accuracy on that dataset for a single feature type at the date of publication of our CVPR paper (Boureau et al., 2010a). Discriminative training of dictionaries with our method on the Caltech-101 dataset has yielded only very little improvement, probably due to the scarcity of training data.

7.3 Conclusion

Supervision usually improves accuracy, however the improvement over the initial parameters obtained with unsupervised training may seem disappointing, compared to the dramatic improvement of switching to a better unsupervised scheme. This is in line with the results from Chapter 3 that the actual dictionary used often matters less than the type of coding.

FUTURE WORK

Preserving closeness relationships in the input, including supervision, and speeding up coding, have all been important themes in this thesis. Here, we briefly revisit these ideas to propose a natural research direction that would tie them all together.

7.4 Learning to predict active sets

The model used in Chapter 5 assigns a neighborhood (e.g., the region around a K-means prototype) to an input; pooling is then performed separately for each neighborhood.

In Chapter 5, we have mainly presented this procedure as a more refined type of pooling, but it can also be viewed as an *expansive coding step* that produces extremely sparse codes. This is true of any type of pooling step that considers more than one pooling region: the "tags" that constitute the addressing information for the compact code (e.g., spatial cell, cluster index) can serve to retrieve either a pool, or a subset of coordinates in a larger sparse code (with pooling then operating on the entire set of codes).

In the setting of Chapter 5, the expanded coding step (sparse coding over the dictionary + expansion into larger-dimensional space) can be constructed as a sparse coding step over a much larger dictionary, where a core subset of codewords (the true underlying dictionary) is replicated P times, if there are P configuration bins. The preclustering step assigns a reduced active set that depends on the configuration bin the input has fallen into.

This view naturally leads to a more general setting, where:

- the codewords are not constrains to be identical from one active set to another,
- the active sets are allowed to overlap and have different sizes,
- the assignment to an active set is performed by any type of classifier, not necessarily a K-means classifier.

The generalized model is illustrated in Fig. (7.1). This opens several avenues: using soft assignments, the classifier can be trained to favor active sets that produce the best code (where the "goodness" of the code is measured by the ℓ_1-regularized loss that is minimized during coding), or in a supervised or semi-supervised way with label information. A coarse-to-fine strategy can be followed, where a coarse decomposition over a small dictionary is first performed, and is used as additional input to the classifier to expand the authorized active set for a finer decomposition; the process can be iterated. This would limit the number of matrix multiplications that have to be performed and potentially allow for gigantic dictionaries to be used very fast, in a spirit similar to (Szlam et al., 2012).

7.5 Connection to structured sparse coding

The coarse-to-fine strategy described above can also be viewed as endowing a dictionary with hierarchical structure. Recent work in structured sparsity (Jenatton et al., 2010) creates a constraint structure where some atoms can be used only if a parent atom is used. The optimization is done jointly, but a greedy algorithm using this structure would

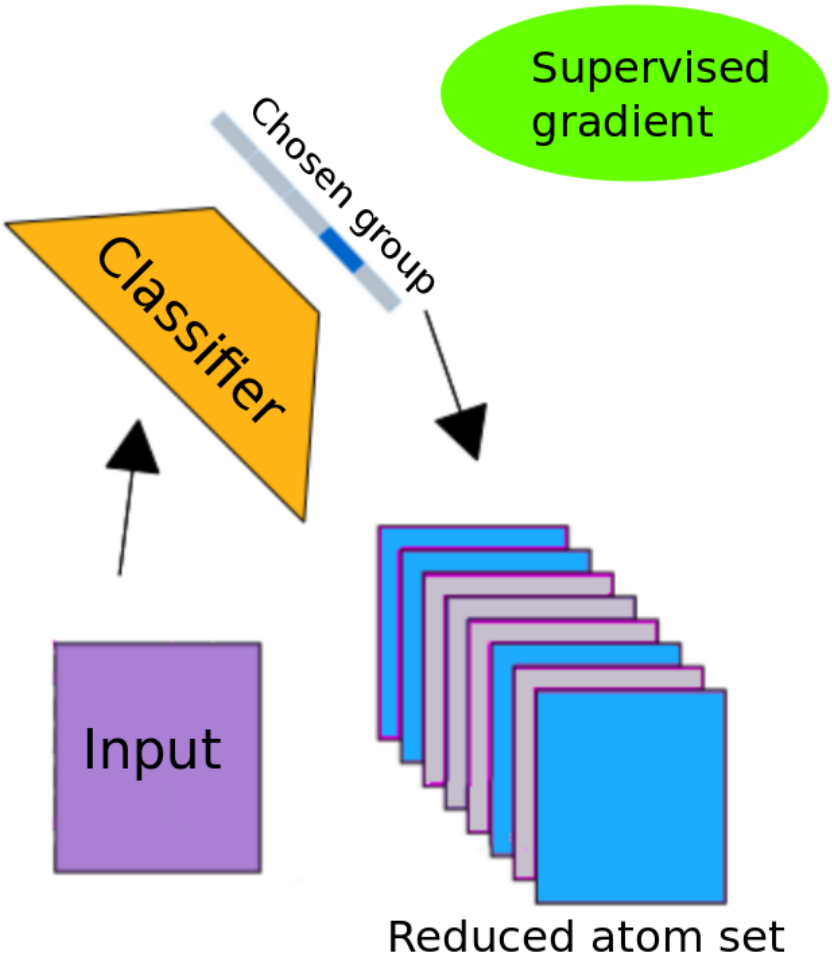


Figure 7.1: Architecture generalizing the model in Chapter 5. The input is first passed to a classifier that predicts a latent class. Each class is associated with a reduced active set of a large dictionary. Coding is then restricted to that active set.

be very similar to the scheme we have outlined: the pattern of activation of the parent atoms is fed to a classifier which decides what children nodes can be used.

Aggregated coding (Jégou et al., 2010) and super-vector coding (Zhou et al., 2010) can also be viewed as realizing this kind of greedy structured sparse coding, over a hybrid dictionary that is composed of P K-means centroids (as parent nodes), and $P * 128$ other atoms which are P replications of the canonical basis of the 128-dimensional input space.

A dictionary adapted to this inference could be trained by specifying active sets at the beginning of training (e.g., by assigning them randomly to hidden classes), much like in previous work using group sparsity or structured sparsity penalties (Hyvärinen and Hoyer, 2001; Kavukcuoglu et al., 2009; Jenatton et al., 2010; Gregor et al., 2011).

CONCLUSION

Throughout this thesis, we have teased apart components of common image recognition systems to gain a better understanding of what makes them work. The generic feature extraction module we have proposed in Chapter 2 crucially pairs a coding and a pooling module, and much of our work has attempted to show that the best performing systems use pairs that complement each other. One example is that rarity of activation of a feature is a strength when combined with the statistical properties of max pooling, as observed in Chapter 3 and analyzed in Chapter 4. Another example is that a refined pooling module can compensate for a coarser coding step, and vice-versa, as seen in Chapter 5. An intriguing question is whether this close partnership extends to the classification kernel; e.g., applying an extremal kernel such as the intersection kernel could have similar effects as using extremal pooling like max pooling. We have proposed macrofeatures in Chapter 3, and shown that a small adjustment in the articulation between the low-level and mid-level layers of feature extraction can lead to reliable performance gains. We have conducted an extensive evaluation of unsupervised and supervised coding modules, combined with several pooling modules and classifier types, in Chapter 3, Chapter 6 and Chapter 7, and have proposed several ways to speed up coding without sacrificing too much recognition accuracy, in Chapter 6. We have presented architectures that are trained convolutionally and perform better than those trained on patches in Chapter 6.

Several outstanding questions would be interesting to pursue. First, it is unclear why

fully trained architectures, while competitive, still cannot outperform more engineered systems. Second, we have examined many aspects of pooling, but not considered the best way to pool across basis functions; knowing what basis functions go together would probably be a very important source of robustness to perturbations. Third, the view of pooling as a statistic extractor would suggest many other types of pooling operators to compare: higher moments such as variance, skewness, kurtosis, or quantile statistics could better characterize the statistics of feature activations over a pool. Feature extraction may still sometimes seem more like a craft than a science, but answering these questions would take us a bit further towards understanding why our models work, and how to make them work better.

APPENDIX

7.6 Proofs for the statistics of max pooling with an exponential distribution

Assume the distribution of the value of a feature for each patch is an exponential distribution with mean $1/\lambda$ and variance $1/\lambda^2$. The corresponding cumulative distribution function is $1 - e^{-\lambda x}$. The cumulative distribution function of the max-pooled feature is $(1 - e^{-\lambda x})^P$. Then the mean and the variance of the distribution of the max-pooled feature are: $\mu_m = \mathcal{H}(P)/\lambda$ and $\sigma_m^2 = 1/\lambda^2 \sum_{l=1}^{P} 1/l(2\mathcal{H}(l) - \mathcal{H}(P))$, respectively, where $\mathcal{H}(k) = \sum_{i=1}^{k} 1/i$ denotes the harmonic series.

PROOF:

- MEAN: The cumulative distribution function of the max-pooled feature is:

$$F(x) \triangleq (1 - \exp(-\lambda x))^P. \tag{7.13}$$

Hence,

$$\mu_m = \int_0^\infty [1 - F(x)]dx = \int_0^\infty 1 - (1 - \exp(-\lambda x))^P dx. \tag{7.14}$$

Changing variable to $u = (1 - \exp(-\lambda x))$:

$$\mu_m = \frac{1}{\lambda} \int_0^1 \frac{1 - u^P}{1 - u} du = \frac{1}{\lambda} f(P), \tag{7.15}$$

where $f_1(P) \triangleq \int_0^1 \frac{1-u^P}{1-u} du$. We have:

$$f_1(0) = 0; \forall P \geq 1, f_1(P) - f_1(P-1) = \int_0^1 u^{P-1} du = \frac{1}{P}. \tag{7.16}$$

Hence, $\mu_m = 1/\lambda \sum_{j=1}^{P} 1/j = \mathcal{H}(P)/\lambda$. □

- VARIANCE: $\sigma_m^2 = \mathbb{E}[X^2] - \mu_m^2$;

$$\mathbb{E}[X^2] = 2\int_0^\infty x[1 - F(x)]dx \tag{7.17}$$

$$= 2\int_0^\infty x.(1 - (1 - \exp(-\lambda x))^P)dx \tag{7.18}$$

$$= \frac{2}{\lambda^2}\int_0^1 \frac{1-u^P}{1-u}(-\log(1-u))du, \tag{7.19}$$

$$\tag{7.20}$$

with the same change of variables as before. Hence:

$$f_2(P) - f_2(P-1) \triangleq \frac{\lambda^2}{2}\left(\mathbb{E}[X^2]_P - \mathbb{E}[X^2]_{P-1}\right) = \int_0^1 u^{P-1}(-\log(1-u))du. \tag{7.21}$$

Integrating by parts with $1 - u^P$ yields:

$$f_2(P) - f_2(P-1) = \frac{1}{P}\int_0^1 \frac{1-u^P}{1-u} du \tag{7.22}$$

$$= \frac{\mathcal{H}(P)}{P}. \tag{7.23}$$

Hence:

$$\sigma_m^2 = \frac{1}{\lambda^2}\sum_{l=1}^{P} \frac{1}{l}(2\mathcal{H}(l) - \mathcal{H}(P)) \tag{7.24}$$

□

7.7 Additional experimental results

7.7.1 Dependency on the regularization hyperparameter of the SVM

Experiments here show results as a function of the regularization (C) hyperparameter of the SVM classifier. As can be seen in Fig. (7.2), performance is stable on a wide range of regularization hyperparameters, but choosing a wrong C can be severely damaging..

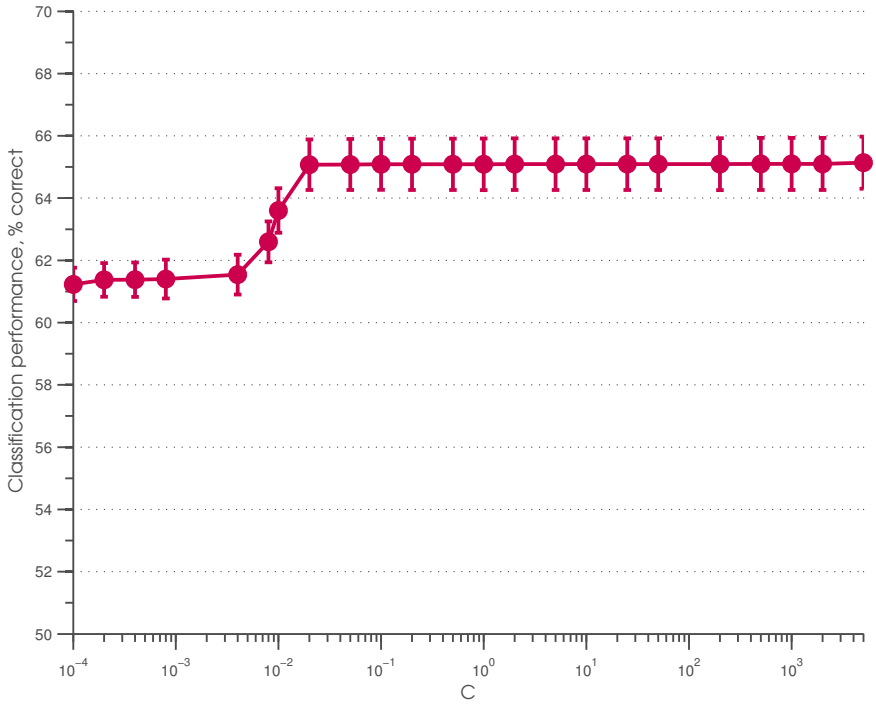


Figure 7.2: Recognition accuracy, Caltech 101 dataset, 15 training examples, sparse coding, max pooling, linear classifier, standard features, when varying the C regularization hyperparameter of the SVM.

7.7.2 Additional results for Chapter 3

Fig. (7.3) shows the same results as Fig. (3.6), but using 15 training images per class instead of 30.

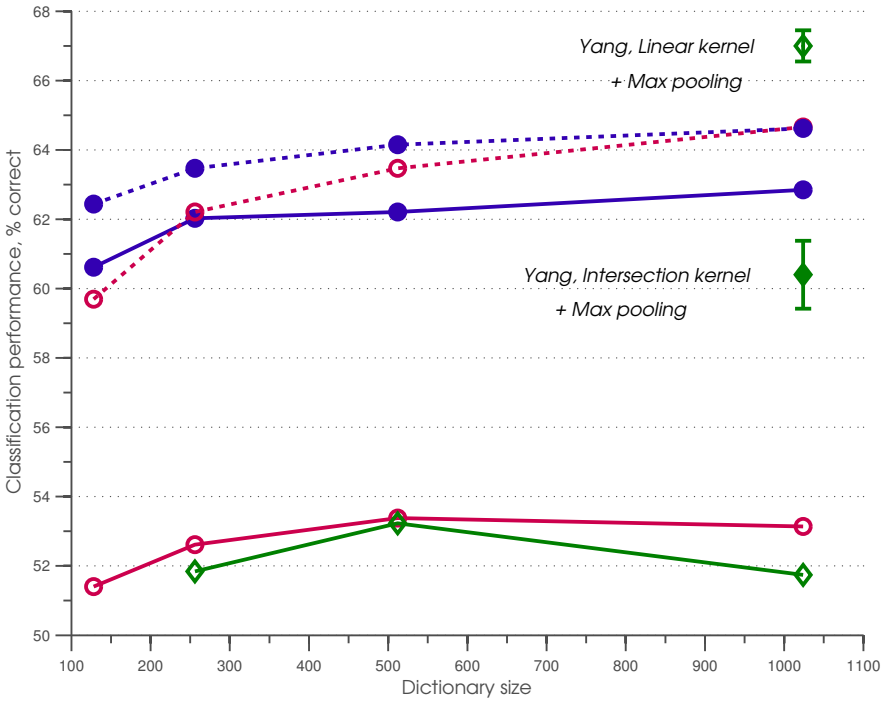


Figure 7.3: Recognition accuracy, Caltech 101 dataset, 15 training examples, with sparse codes and different combinations of pooling and kernels. Dotted lines: max pooling. Solid lines: average pooling. Closed symbols, blue: intersection kernel. Open symbols, red: linear kernel. Green: Yang et al.'s results (Yang et al., 2009b).

Caltech-101	$p = 1$	$p = 4$	$p = 16$	$p = 64$
$k = 4$	28.0 ± 0.7	38.5 ± 1.0	53.9 ± 1.0	62.8 ± 0.8
$k = 16$	53.1 ± 1.0	58.3 ± 1.1	63.2 ± 1.0	68.6 ± 1.0
$k = 64$	62.8 ± 1.0	66.7 ± 1.1	69.7 ± 0.8	73.0 ± 0.6
Scenes	$p = 1$	$p = 4$	$p = 16$	$p = 64$
$k = 4$	40.6 ± 0.7	56.6 ± 0.7	64.5 ± 0.6	70.1 ± 0.8
$k = 16$	63.3 ± 0.8	69.3 ± 0.7	74.0 ± 0.6	76.0 ± 0.6
$k = 64$	72.6 ± 0.6	76.6 ± 1.0	78.9 ± 0.5	79.2 ± 0.5
Caltech-256	$p = 1$	$p = 4$	$p = 16$	$p = 64$
$k = 4$	6.9 ± 0.3	11.7 ± 0.4	17.4 ± 0.3	21.0 ± 0.3
$k = 16$	16.5 ± 0.3	20.9 ± 0.4	24.8 ± 0.3	26.5 ± 0.3
$k = 64$	23.0 ± 0.3	26.5 ± 0.3	29.8 ± 0.3	–

Table 7.4: Recognition accuracy for smaller dictionaries, as a function of K: size of the codebook for sparse coding, and P: number of clusters for pooling. Precluster-ing needs larger final global image representations to outperform richer dictionaries. Macrofeatures used for Caltech-101 and Scenes with image resizing, standard features with full-size images for Caltech-256.

7.7.3 Additional results for Chapter 5

This section contains numerical results of experiments plotted in the chapter, and results from the Caltech-256 and Scenes datasets.

p	1	4	16
$k = 256$	32.9 ± 0.4	34.5 ± 0.5	38.0 ± 0.6
$k = 1024$	39.7 ± 0.3	39.2 ± 0.5	40.8 ± 0.6
$k = 256$	32.3 ± 0.8	34.9 ± 0.5	38.0 ± 0.5
$k = 1024$	38.1 ± 0.6	39.8 ± 0.7	41.6 ± 0.6

Table 7.5: Recognition accuracy on Caltech 256, 30 training examples, as a function of K: size of the codebook for sparse coding, and P: number of clusters extracted on the input data. Macrofeatures extracted every 4 pixels. Top two rows: no resizing of the image. Bottom two rows: resized so that maximal dimension is ≤ 300 pixels.

C101	$k = 4$	$k = 16$	$k = 64$	$k = 256$	$k = 1024$
Single	38.5 ± 1.0	58.3 ± 1.1	66.7 ± 1.1	72.6 ± 1.0	76.0 ± 1.2
Sep	45.9 ± 1.1	60.0 ± 1.0	68.2 ± 1.3	73.4 ± 0.9	76.7 ± 1.0
Scenes	$k = 4$	$k = 16$	$k = 64$	$k = 256$	$k = 1024$
Single	56.6 ± 0.7	69.3 ± 0.7	76.6 ± 1.0	81.7 ± 0.5	83.5 ± 0.8
Sep	61.5 ± 0.5	70.6 ± 0.8	78.2 ± 0.7	81.8 ± 0.7	83.7 ± 0.8

Table 7.6: Recognition accuracy on Caltech-101 and Scenes, according to whether a separate dictionary is learned for each of $P = 4$ clusters. Single: shared dictionary. Sep: one dictionary per cluster. Dictionary size K.

		Caltech 30 tr.	Scenes
Pre,	$P = 1$	70.5 ± 0.8	79.2 ± 0.7
	$P = 4$	72.6 ± 1.0	81.7 ± 0.5
	$P = 16$	74.0 ± 1.0	82.0 ± 0.7
	$P = 64$	75.0 ± 0.8	81.4 ± 0.4
	$P = 128$	75.5 ± 0.8	81.0 ± 0.3
	$P = 256$	75.1 ± 1.0	$-$
	$P = 512$	74.5 ± 0.7	$-$
	$P = 1024$	73.8 ± 0.8	$-$
	$P = 1 + 16$	74.2 ± 1.1	81.5 ± 0.8
	$P = 1 + 64$	75.6 ± 0.6	81.9 ± 0.7
Post,	$P = 4$	72.4 ± 1.2	79.6 ± 0.8
	$P = 16$	75.1 ± 0.8	80.9 ± 0.6
	$P = 64$	76.4 ± 0.8	81.1 ± 0.6
	$P = 128$	$\mathbf{76.7 \pm 0.8}$	81.1 ± 0.5
	$P = 256$	75.9 ± 0.8	$-$
	$P = 512$	75.2 ± 0.8	$-$
	$P = 1024$	74.2 ± 0.6	$-$

Table 7.7: Accuracy on Caltech-101 and Scenes as a function of whether clustering is performed before (Pre) or after (Post) the encoding, for a dictionary size $K = 256$. P: number of configuration space bins.

		Caltech 30 tr.	Scenes
Pre,	$P = 1$	75.6 ± 0.9	82.7 ± 0.7
	$P = 4$	76.0 ± 1.2	83.5 ± 0.8
	$P = 16$	76.3 ± 1.1	82.8 ± 0.8
	$P = 64$	76.2 ± 0.8	81.8 ± 0.7
	$P = 128$	$-$	80.9 ± 0.7
	$P = 1 + 16$	$\mathbf{76.9 \pm 1.0}$	83.3 ± 1.0
	$P = 1 + 64$	$\mathbf{77.3 \pm 0.6}$	83.1 ± 0.7
Post,	$P = 4$	$75.8 \pm 1.5.$	82.9 ± 0.6
	$P = 16$	$\mathbf{77.0 \pm 0.8}$	82.9 ± 0.5
	$P = 64$	$\mathbf{77.1 \pm 0.7}$	82.4 ± 0.7
	$P = 128$	76.9 ± 0.5	82.0 ± 0.7
	$P = 256$	75.7 ± 0.8	$-$

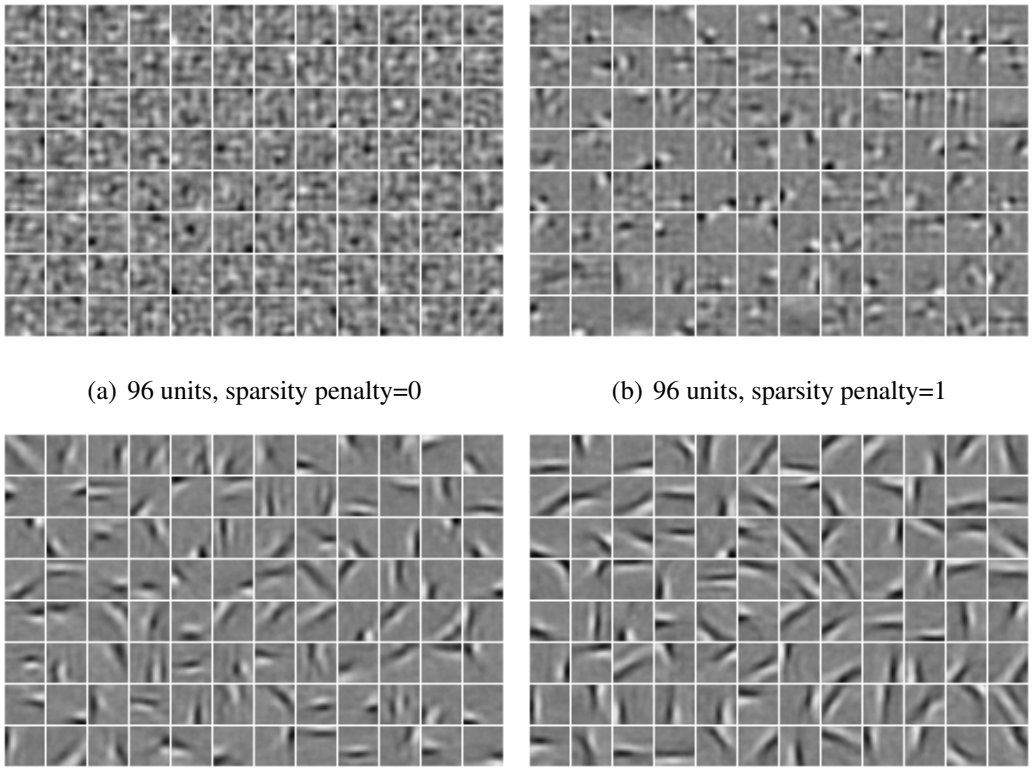
Table 7.8: Accuracy on Caltech-101 and Scenes as a function of whether clustering is performed before (Pre) or after (Post) the encoding, for a dictionary size $K = 1024$. P: number of configuration space bins.

w_i	$k = 4$	$k = 16$	$k = 64$	$k = 256$
Caltech-101, average pooling				
1	44.9 ± 0.8	54.8 ± 0.8	60.7 ± 0.9	64.1 ± 0.9
$\sqrt{n_i/n}$	49.7 ± 0.9	56.6 ± 0.8	62.3 ± 1.1	65.7 ± 1.2
n_i/n	44.6 ± 0.8	51.5 ± 0.9	58.1 ± 1.2	62.8 ± 1.3
Scenes, average pooling				
1	56.4 ± 0.6	66.3 ± 1.0	72.0 ± 0.6	74.5 ± 0.6
$\sqrt{n_i/n}$	62.6 ± 0.8	68.5 ± 0.8	72.4 ± 0.4	74.6 ± 0.8
n_i/n	60.9 ± 1.1	65.7 ± 0.8	70.2 ± 0.4	72.5 ± 0.9
Caltech-256, average pooling				
1	12.6 ± 0.2	17.9 ± 0.3	19.9 ± 0.4	–
$\sqrt{n_i/n}$	16.8 ± 0.4	20.4 ± 0.4	21.9 ± 0.3	–
n_i/n	14.8 ± 0.3	17.8 ± 0.4	20.2 ± 0.3	–

Table 7.9: Recognition accuracy on Caltech-101, Scenes, and Caltech-256, for different cluster weighting schemes, with average pooling, for $P = 16$ clusters, and dictionary size K. Macrofeatures on resized images are used for Caltech-101 and Scenes, standard features on full-size images for Caltech-256.

w_i	$k = 4$	$k = 16$	$k = 64$	$k = 256$
Caltech-101, max pooling				
1	53.9 ± 1.0	63.2 ± 1.0	69.7 ± 0.8	74.0 ± 1.0
$\sqrt{n_i/n}$	53.9 ± 1.2	61.6 ± 0.6	66.7 ± 0.8	70.9 ± 1.0
n_i/n	47.0 ± 1.1	54.9 ± 1.1	60.9 ± 0.8	65.5 ± 0.9
Scenes, max pooling				
1	64.5 ± 0.6	74.0 ± 0.6	78.9 ± 0.5	81.5 ± 0.8
$\sqrt{n_i/n}$	66.1 ± 1.0	72.7 ± 0.8	77.2 ± 0.6	79.5 ± 0.9
n_i/n	63.4 ± 0.9	69.2 ± 0.7	74.2 ± 0.8	76.9 ± 0.8
Caltech-256, max pooling				
1	17.4 ± 0.3	24.8 ± 0.3	29.8 ± 0.3	–
$\sqrt{n_i/n}$	19.3 ± 0.4	24.5 ± 0.3	28.2 ± 0.3	–
n_i/n	16.4 ± 0.3	20.7 ± 0.2	24.0 ± 0.3	–

Table 7.10: Recognition accuracy on Caltech-101, Scenes, and Caltech-256, for different cluster weighting schemes, with max pooling, for $P = 16$ clusters, and dictionary size K. Macrofeatures on resized images are used for Caltech-101 and Scenes, standard features on full-size images for Caltech-256.



(a) 96 units, sparsity penalty=0

(b) 96 units, sparsity penalty=1

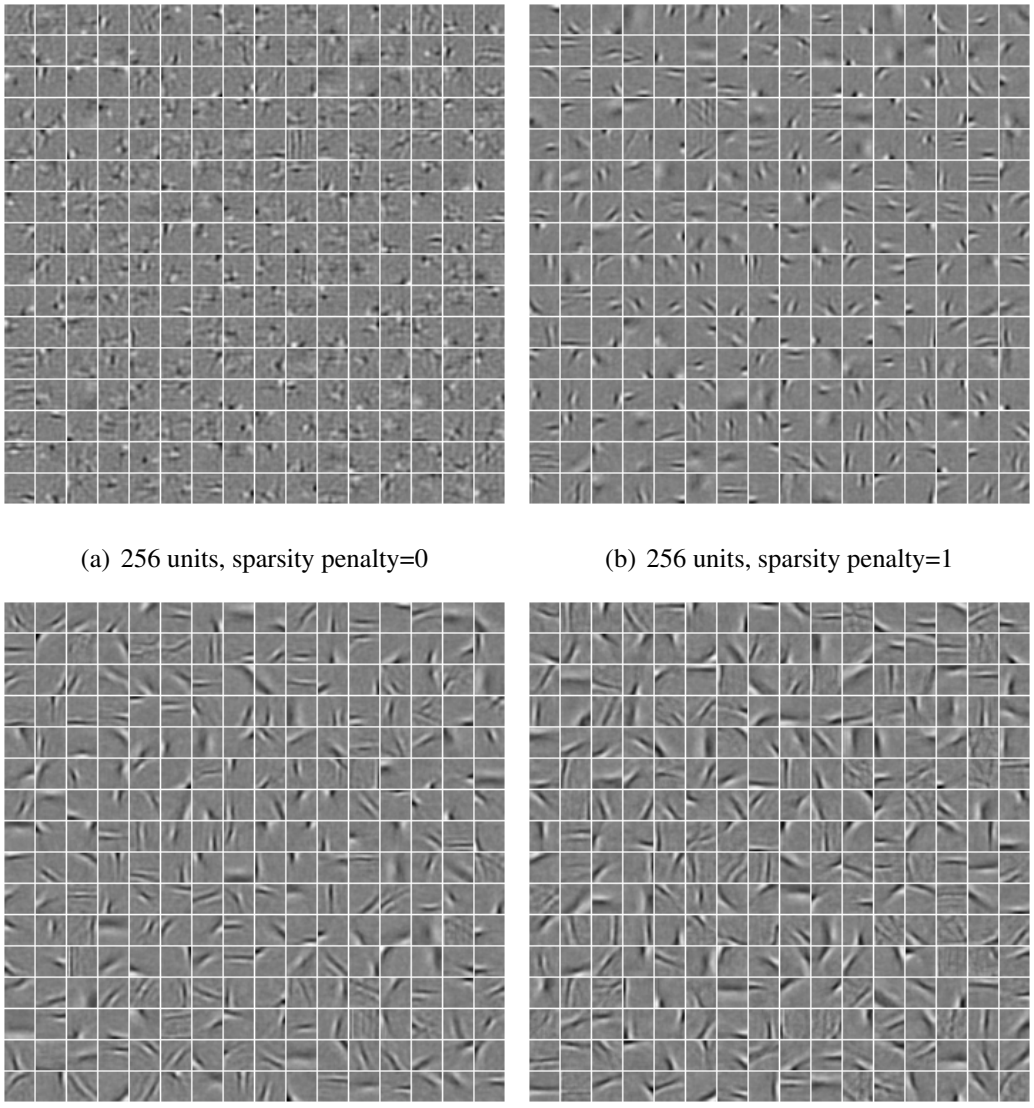
(c) 96 units, sparsity penalty=5

(d) 96 units, sparsity penalty=25

Figure 7.4: 96 RBM hidden units trained with increasingly weighted sparsity penalty over 21×21 patches of the Caltech-101 dataset.

7.7.4 Additional filter images for Sec. (6.3.1)

Fig. (7.4), Fig. (7.5), Fig. (7.6) and Fig. (7.7) show 96, 256, and 512 RBM hidden units, trained over 21×21 patches of the Caltech-101 dataset, with increasingly weighted sparsity.



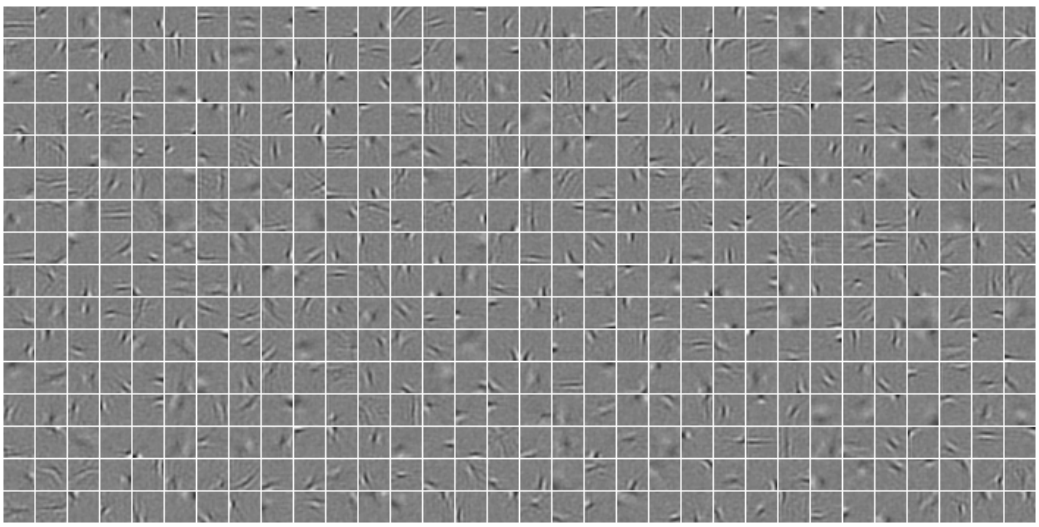
(a) 256 units, sparsity penalty=0

(b) 256 units, sparsity penalty=1

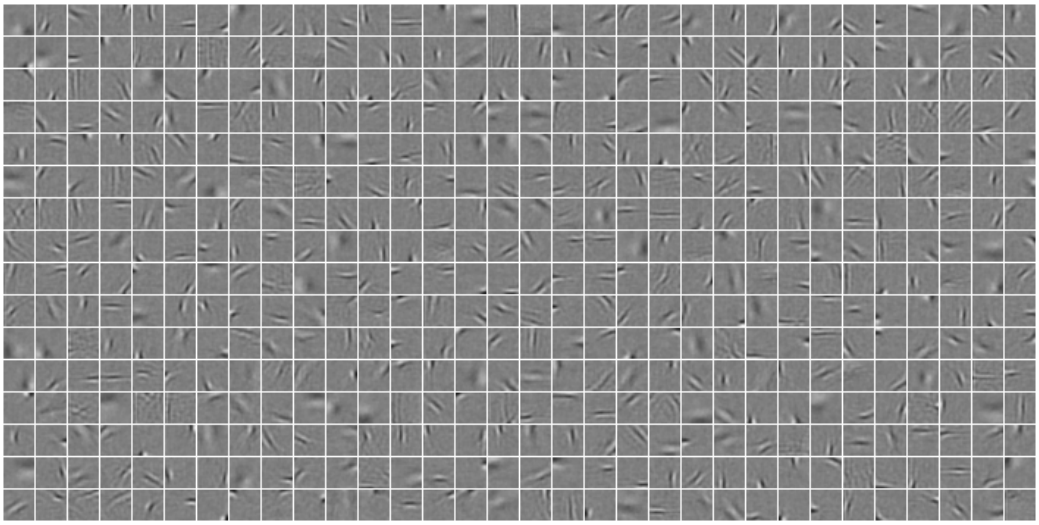
(c) 256 units, sparsity penalty=5

(d) 256 units, sparsity penalty=25

Figure 7.5: 256 RBM hidden units trained with increasingly weighted sparsity penalty over 21×21 patches of the Caltech-101 dataset.

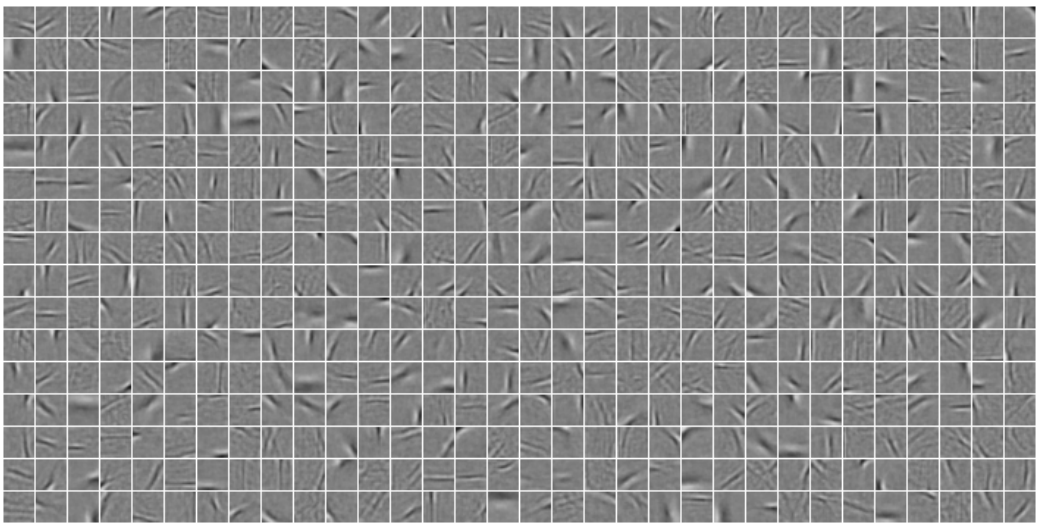


(a) 512 units, sparsity penalty=0

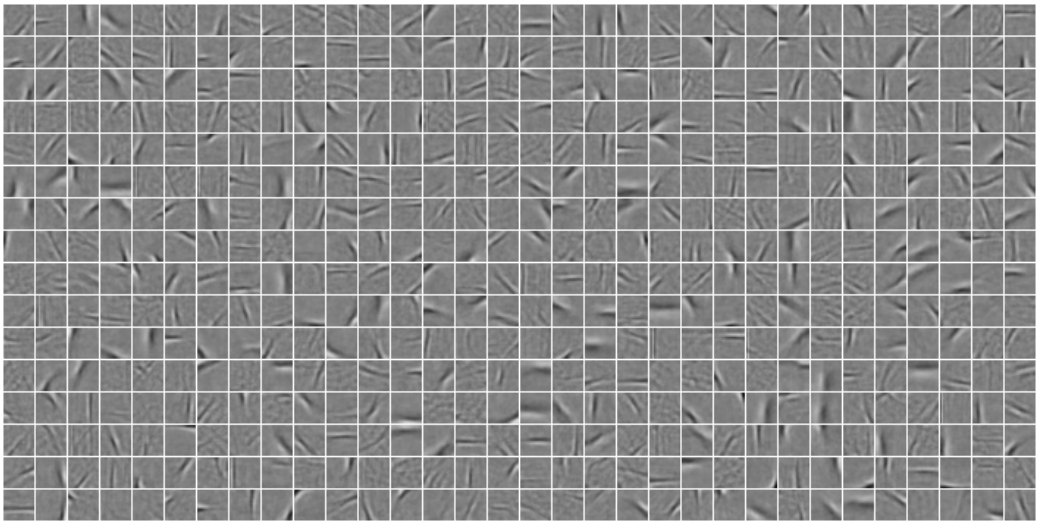


(b) 512 units, sparsity penalty=1

Figure 7.6: 512 RBM hidden units trained with increasingly weighted sparsity penalty over 21×21 patches of the Caltech-101 dataset.



(a) 512 units, sparsity penalty=5



(b) 512 units, sparsity penalty=25

Figure 7.7: 512 RBM hidden units trained with increasingly weighted sparsity penalty over 21×21 patches of the Caltech-101 dataset.

BIBLIOGRAPHY

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

Ahmed, N., Natarajan, T., and Rao, K. (1974). Discrete cosine transform. *Computers, IEEE Transactions on*, 100(1):90–93.

Bach, F. and Harchaoui, Z. (2008). DIFFRAC: a discriminative and flexible framework for clustering. *Advances in Neural Information Processing Systems*, 20.

Beck, A. and Teboulle, M. (2009). A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM J. Img. Sci.*, 2(1):183–202.

Bengio, Y. (2007). Learning deep architectures for AI. Technical Report 1312, Dept. IRO, Universite de Montreal.

Bengio, Y. and LeCun, Y. (2007). Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. D. and Weston, J., editors, *Large-Scale Kernel Machines*. MIT Press.

Boiman, O., Rehovot, I., Shechtman, E., and Irani, M. (2008). In Defense of Nearest-Neighbor Based Image Classification. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008. CVPR 2008.*

Bottou, L. (1998). Online Algorithms and Stochastic Approximations. In Saad, D., editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK.

Boureau, Y.-L., Bach, F., LeCun, Y., and Ponce, J. (2010a). Learning Mid-Level Features for Recognition. In *CVPR'10*. IEEE.

Boureau, Y.-L., Le Roux, N., Bach, F., Ponce, J., and LeCun, Y. (2011). Ask the locals: multi-way local pooling for image recognition. In *Proc. International Conference on Computer Vision (ICCV'11)*. IEEE.

Boureau, Y.-L., Ponce, J., and LeCun, Y. (2010b). A theoretical analysis of feature pooling in vision algorithms. In *Proc. International Conference on Machine learning (ICML'10)*.

Bruckstein, A. M. and Cover, T. M. (1985). Monotonicity of linear separability under translation. *IEEE TRANS. PATTERN ANAL. MACH. INTELLIG.*, 7(3):355–357.

Buades, A., Coll, B., and Morel, J. M. (2005). A non-local algorithm for image denoising. In *CVPR*.

Carandini, M. (2006). What simple and complex cells compute. *J Physiol*, 577(Pt 2):463–466.

Chang, C.-C. and Lin, C.-J. (2001). *{LIBSVM}: a library for support vector machines*.

Chen, S. S., Donoho, D. L., and Saunders, M. A. (1999). Atomic Decomposition by Basis Pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61.

Coates, A., Lee, H., and Ng, A. (2011). An analysis of single-layer networks in unsupervised feature learning. In *AISTATS 2011*.

Coates, A. and Ng, A. Y. (2011). The importance of encoding versus training with sparse coding and vector quantization. In *International Conference on Machine Learning*, volume 8, page 10.

Collobert, R. and Weston, J. (2008). A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *International Conference on Machine Learning, {ICML}*.

Dabov, K., Foi, A., Katkovnik, V., and Egiazarian, K. (2006). Image denoising with block-matching and {3D} filtering. In *Proc. SPIE Electronic Imaging: Algorithms and Systems V*, volume 6064, San Jose, CA, USA.

Dalal, N. and Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, volume 2, pages 886–893.

Donoho, D. L. (2006). Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306.

Duchenne, O., Joulin, A., and Ponce, J. (2011). A Graph-Matching Kernel for Object Categorization. In *Proceedings of the International Conference in Computer Vision (ICCV)*.

Efron, B., Hastie, T., and andRobert Tibshirani, I. J. (2004). Least angle regression. *Annals of Statistics*, 32(2).

Elad, M. and Aharon, M. (2006). Image Denoising Via Learned Dictionaries and Sparse representation. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 895–900, Washington, DC, USA. IEEE Computer Society.

Fei-Fei, L., Fergus, R., and Perona, P. (2004). Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories. In *CVPR Workshop on Generative Model-Based Vision*.

Fei-Fei, L. and Perona, P. (2005). A Bayesian Hierarchical Model for Learning Natural Scene Categories. In *CVPR*.

Freeman, W., Adelson, E., of Technology. Media Laboratory. Vision, M. I., and Group, M. (1991). The design and use of steerable filters. *IEEE Transactions on Pattern analysis and machine intelligence*, 13(9):891–906.

Friedman, J., Hastie, T., Höfling, H., and Tibshirani, R. (2007). Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332.

Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1.

Fukushima, K. and Miyake, S. (1982). Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*.

Gao, S., Tsang, I. W. H., Chia, L. T., and Zhao, P. (2010). Local features are not lonely–Laplacian sparse coding for image classification. In *CVPR*.

Gehler, P. and Nowozin, S. (2009). On Feature Combination for Multiclass Object Classification. In *ICCV*.

Goldberger, J., Roweis, S. T., Hinton, G. E., and Salakhutdinov, R. (2004). Neighbourhood Components Analysis. In *NIPS*.

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

Grauman, K. and Darrell, T. (2005). The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features. In *ICCV*.

Gregor, K. and LeCun, Y. (2010). Learning Fast Approximations of Sparse Coding. In *Proc. International Conference on Machine learning (ICML'10)*.

Gregor, K., Szlam, A., and LeCun, Y. (2011). Structured sparse coding via lateral inhibition. In *Advances in Neural Information Processing Systems (NIPS 2011)*, volume 24.

Griffin, G., Holub, A., and Perona, P. (2007). Caltech-256 Object Category Dataset. Technical Report UCB/CSD-04-1366, California Institute of Technology.

Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality Reduction by Learning an Invariant Mapping. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'06)*. IEEE Press.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800.

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

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

Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J Physiol*, 160:106–154.

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

Jain, P., Kulis, B., and Grauman, K. (2008). Fast image search for learned metrics. In *CVPR*, pages 1–8.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the Best Multi-Stage Architecture for Object Recognition? In *(ICCV'09)*. IEEE.

Jégou, H., Douze, M., Schmid, C., and Pérez, P. (2010). Aggregating local descriptors into a compact image representation. In *CVPR*.

Jenatton, R., Mairal, J., Obozinski, G., and Bach, F. (2010). Proximal methods for sparse hierarchical dictionary learning. In *ICML*.

Kanan, C. and Cottrell, G. (2010). Robust classification of objects, faces, and flowers using natural image statistics. In *CVPR*.

Kavukcuoglu, K., Ranzato, M., Fergus, R., and LeCun, Y. (2009). Learning Invariant Features through Topographic Filter Maps. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR'09)*. IEEE.

Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2008). Fast inference in sparse coding algorithms with applications to object recognition. Technical report, Computational and Biological Learning Lab, Courant Institute, NYU. Tech Report CBLL-TR-2008-12-01.

Kavukcuoglu, K., Sermanet, P., Boureau, Y.-L., Gregor, K., Mathieu, M., and LeCun, Y. (2010). Learning Convolutional Feature Hierachies for Visual Recognition. In *Advances in Neural Information Processing Systems (NIPS 2010)*.

Kim, J. and Grauman, K. (2010). Asymmetric Region-to-Image Matching for Comparing Images with Generic Object Categories. In *CVPR*.

Koenderink, J. J. and Van Doorn, A. J. (1999). The Structure of Locally Orderless Images. *IJCV*, 31(2/3):159–168.

Lazebnik, S. and Raginsky, M. (2008). Supervised Learning of Quantizer Codebooks by Information Loss Minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21.

Lazebnik, S., Schmid, C., and Ponce, J. (2006). Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In *Proc. IEEE Conf. Comp. Vision Patt. Recog.*, volume II, pages 2169–2178.

LeCun, Y. and Bengio, Y. (1995). Convolutional Networks for Images, Speech, and Time-Series. In Arbib, M. A., editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In Touretzky, D., editor, *Advances in Neural Information Processing Systems 2 (NIPS*89)*, Denver, CO. Morgan Kaufman.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

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

Lee, H., Battle, A., Raina, R., and Ng, A. Y. (2006). Efficient sparse coding algorithms. In Schölkopf, B., Platt, J. C., and Hoffman, T., editors, *NIPS*, pages 801–808. MIT Press.

Lee, H., Chaitanya, E., and Ng, A. Y. (2007). Sparse deep belief net model for visual area V2. In *Advances in Neural Information Processing Systems*.

Lee, H., Grosse, R., Ranganath, R., and Ng, A. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *ICML*.

Li, Y. and Osher, S. (2009). Coordinate descent optimization for l1 minimization with application to compressed sensing; a greedy algorithm. *Inverse Problems and Imaging*, 3(3):487–503.

Lloyd, S. P. (1982). Least squares quantization in {PCM}. {*IEEE*} *Trans. Inf. Theory*, 28:129–137.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *Int. J. of Comp. Vision*, 60(4):91–110.

Lowe, D. G. (1999). Object Recognition from Local Scale-Invariant Features. In *ICCV*.

Lyu, S. and Simoncelli, E. P. (2008). Nonlinear image representation using divisive normalization. In *CVPR'08*. IEEE Computer Society.

Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2009a). Online Dictionary Learning for Sparse Coding. In *Proceedings of the 26th international conference on Machine learning*.

Mairal, J., Bach, F., Ponce, J., Sapiro, G., and Zisserman, A. (2009b). Non-local sparse models for image restoration. In *CVPR 2009*.

Mairal, J., Bach, F., Ponce, J., Sapiro, G., and Zisserman, A. (2009c). Supervised Dictionary Learning. *Advances in Neural Information Processing Systems*, 21.

Mallat, S. (1999). *A wavelet tour of signal processing*. Academic Press.

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

Martin, D., Fowlkes, C., Tal, D., and Malik, J. (2001). A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics. In *ICCV'01*, volume 2, pages 416–423.

Mikolajczyk, K. and Schmid, C. (2005). A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 27(10):1615–1630.

Minka, T. P. (2001). Expectation Propagation for approximate Bayesian inference. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence table of contents*, pages 362–369. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.

Olshausen, B. and Field, D. (2005). How close are we to understanding v1? *Neural computation*, 17(8):1665–1699.

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

Pinto, N., Cox, D. D., and DiCarlo, J. J. (2008). Why is real-world visual object recognition hard. *PLoS Computational Biology*, 4(1):151–156.

Raina, R., Battle, A., Lee, H., Packer, B., and Ng, A. (2007). Self-taught learning: Transfer learning from unlabeled data. pages 759–766.

Ranzato, M., Boureau, Y., Chopra, S., and LeCun, Y. (2007a). A Unified Energy-Based Framework for Unsupervised Learning. In *Proc. Conference on AI and Statistics (AI-Stats)*.

Ranzato, M., Boureau, Y.-L., and LeCun, Y. (2007b). Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems (NIPS 2007)*.

Ranzato, M., Huang, F.-J., Boureau, Y.-L., and LeCun, Y. (2007c). Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press.

Ranzato, M., Poultney, C., Chopra, S., and LeCun, Y. (2006). Efficient Learning of Sparse Representations with an Energy-Based Model. In *NIPS 2006*. MIT Press.

Saul, L. K. and Roweis, S. T. (2003). Think globally, fit locally: unsupervised learning of low dimensional manifolds. *JMLR*, 4:119–155.

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

Serre, T., Wolf, L., and Poggio, T. (2005). Object Recognition with Features Inspired by Visual Cortex. In *CVPR*.

Simoncelli, E. P., Freeman, W. T., Adelson, E. H., and Heeger, D. J. (1998). Shiftable multi-scale transforms. *IEEE Trans. Information Theory*, 38(2):587–607.

Sivic, J. and Zisserman, A. (2003). Video Google: A text retrieval approach to object matching in videos. In *Proceedings of the International Conference on Computer Vision*, volume 2, pages 1470–1477.

SPAMS (2012 (accessed April 2012)). SPArse Modeling Software. `http://www.di.ens.fr/willow/SPAMS/`.

Szlam, A., Gregor, K., and LeCun, Y. (2012). Fast approximations to structured sparse coding and applications to object classification. Technical report.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8:257—-277.

Tibshirani, R. et al. (1997). The lasso method for variable selection in the cox model. *Statistics in medicine*, 16(4):385–395.

Todorovic, S. and Ahuja, N. (2008). Learning subcategory relevances for category recognition. In *CVPR*.

van Gemert, J. C., Veenman, C. J., Smeulders, A. W. M., and Geusebroek, J. M. (2010). Visual Word Ambiguity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (in press).

Vedaldi, A., Gulshan, V., Varma, M., and Zisserman, A. (2009). Multiple Kernels for Object Detection. In *Proc. Int. Conf. Comp. Vision*.

VGG Results URL (2012 (accessed April 2012)). `http://www.robots.ox.ac.uk/~vgg/software/MKL/`.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *ICML '08*.

Viola, P. and Jones, M. J. (2004). Robust Real-Time Face Detection. *Int. J. of Comp. Vision*, 57(2):137–154.

Wang, J., Yang, J., Yu, K., Lv, F., Huang, T., and Gong, Y. (2010). Locality-constrained linear coding for image classification. In *CVPR*.

Weston, J., Rattle, F., and Collobert, R. (2008). Deep Learning via Semi-Supervised Embedding. In *International Conference on Machine Learning, {ICML}*.

Winder, S. and Brown, M. (2007). Learning local image descriptors. In *CVPR*.

Winn, J., Criminisi, A., and Minka, T. (2005). Object categorization by learned universal visual dictionary. In *Tenth IEEE International Conference on Computer Vision, 2005. ICCV 2005*, volume 2.

Wright, J., Yang, A., Ganesh, A., Sastry, S., and Ma, Y. (2009). Robust face recognition via sparse representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(2):210–227.

Yang, J., Li, Y., Tian, Y., Duan, L., and Gao, W. (2009a). Group-Sensitive Multiple Kernel Learning for Object Categorization. In *ICCV*.

Yang, J., Wright, J., Huang, T., and Ma, Y. (2008). Image super-resolution as sparse representation of raw image patches. pages 1–8.

Yang, J., Yu, K., Gong, Y., and Huang, T. (2009b). Linear Spatial Pyramid Matching Using Sparse Coding for Image Classification. In *IEEE Conference on Computer Vision and Pattern Recognition, 2009. CVPR 2009*.

Yang, J., Yu, K., and Huang, T. (2010). Efficient Highly Over-Complete Sparse Coding using a Mixture Model. *ECCV*.

Yu, K., Zhang, T., and Gong, Y. (2009). Nonlinear learning using local coordinate coding. *NIPS*.

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

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

Zhang, H., Berg, A. C., Maire, M., and Malik, J. (2006). {SVM}-{KNN}: Discriminative Nearest Neighbor Classification for Visual Category Recognition. In *CVPR*, pages II: 2126—-2136.

Zhang, J. G., Marszalek, M., Lazebnik, S., and Schmid, C. (2007). Local Features and Kernels for Classification of Texture and Object Categories: {A} Comprehensive Study. *International Journal of Computer Vision*, 73(2):213–238.

Zhou, X., Yu, K., Zhang, T., and Huang, T. (2010). Image Classification using Super-Vector Coding of Local Image Descriptors. *ECCV*.

Zhou, X., Zhuang, X. D., Tang, H., Johnson, M. H., and Huang, T. S. (2008). A novel Gaussianized vector representation for natural scene categorization. In *ICPR*, pages 1–4.