Learning Representations for Computer Vision Tasks

Thesis submitted in partial fulfillment
of the requirements for the degree of

MS by Research
in
Computer Science

by

Siddhartha Chandra
200702049
siddhartha.chandra@research.iiit.ac.in

CENTER FOR VISUAL INFORMATION TECHNOLOGY
International Institute of Information Technology
Hyderabad - 500 032, INDIA
November 2012

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Learning Representations for Computer Vision Tasks" by Siddhartha Chandra, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Prof. C. V. Jawahar

To My Family.

Acknowledgments

I owe this work to the two men who mentored me through my years at CVIT. First of all I would like to thank my research advisor Prof. C. V. Jawahar who I am indebted for life for his presence, keen interest in my research and able guidance. He served both as an inspiration and a goading force throughout my stay at CVIT. Secondly, I was utterly fortunate to have a very involved mentor in Dr. Shailesh Kumar whose expertise in this field of research and valuable lessons in life kept me going through the tough times. Dr. Kumar leads by example, and has had an overwhelming presence in my research career.

I am thankful to my peers at CVIT who have been my friends, critics, mentors, pillars of support in times of distress, and a constant source of inspiration - Vinay, Abhinav, Harshit, Parikshit, Mayank, Shrikant, Yashaswi, Omkar, Srijan, Vempati, Chandrashekar, Ankit, Raman, Anand, Nagendar and others. I am grateful to Mr. R. S. Satyanarayana whose administrative support was indispensable at times. I would like to thank Rajan, Nandini and Phani whose presence at CVIT has helped students in more ways than one. Many thanks to Prof. P. J. Narayan, Prof. Jayanthi and Prof. Anoop for their encouraging presence and for providing an environment conducive to learning of the finest quality at CVIT.

I am thankful also to my peers and mentors at the Visual Geometry Group, University of Oxford. I am grateful to Prof. Andrew Zisserman, Dr. Marcin Marszalek, and Dr. Andrea Vedaldi for their - as AZ would put it - *hunky-dory* guidance through my years of association with VGG. I am thankful to my peers at VGG, Relja, Yusuf, Amr, Varun, Arpit, Marco, and others who have brainstormed with me on different problems at various points in time.

I would like to thank my friends, Sankalp, Chaitanya, Gautam, Ashish, Manish, Mihir, Manan, Pulkit, and others for always believing in me, and making my stay at IIIT worthwhile.

Finally, I would like to acknowledge the contribution of my parents and my sister to my academic and research career. Their unconditional love, support and belief in my abilities have played a pivotal role in shaping the course of my career, and my life. I would also like to thank all the members of my extended family for being there whenever I needed them.

Many thanks to everyone else who affected my life in any way, and wasn't acknowledged personally above.

Preface

A picture of the sky reminds a skilled weather man of three different types of clouds. His four year old son, who lacks the intellectual capacity and the meteoroligical insight of his father, looks at the same picture and claims he sees the shape of a horse in the clouds. *A picture is worth a thousand words.* The information in an image is limited only by the imagination of the observer.

Despite the recent advances in artificial intelligence, computers as an imaginative species remain far inferior to their inventors. What we see as images, computers look at as arrays of numbers. Research in computer vision is dedicated towards bridging the gap between these numbers (*pixel intensity values*) and human vision.

Computer vision researchers have endeavoured for decades to accomplish this herculean task. Often posed as a machine learning problem, solving computer vision involves *the design and development of algorithms that allow computers to evolve behaviors based on empirical data* [1]. A learner takes advantage of examples (data) to capture characteristics of interest of their unknown underlying probability distribution. It then uses this knowledge to drive decisions from the data.

In simpler terms, our solution seeks to understand patterns in the world to teach computers about the world. The method thus involves (a) Observing the world, (b) Developing models that match observations, (c) Teaching the computer to learn these models, and (d) Applying the models learnt by the computer to the world. Observing the world involves collecting data that we use to learn. Developing models that match observations requires us to devise good representations of the world (the collected data) and inventing algorithms that serve our purpose. Teaching the computer involves writing machine readable code that executes our algorithm. Finally, we use the computer program to seek solutions to our problem. A crucial component of the approach is *how we represent the world*. This happens to be the focus of my research work.

In most computer vision tasks our *world* is a set of images or videos[1]. As stated above, a computer sees these as arrays of numbers. A human, when presented with these numbers, may fail to make any sense out of them. To perceive anything that appeals to his visual senses, he needs a tool that draws the picture. Thus, all the information needed to conjure up the image is in the numbers; still, these numbers are useful only when they are presented to the observer in a form the observer can appreciate. A good representation scheme does just this. This principle is the theme of this thesis.

[1] though it is not limited to being a set of images or videos (figure 2.5)

Abstract

Learning representations for computer vision tasks has been the holy grail for the vision community for long. Research in computer vision is dedicated towards developing machines that understand image data, which takes a variety of forms such as images, video sequences, views from multiple cameras, high dimensional data from medical scanners and so on. Good representations of the data intend to discover the hidden structure in it; better insights into the nature of the data can help choose or create better features, learn better similarity measures between data points, build better predictive and descriptive models, and ultimately drive better decisions from data. Research into this field has shown that good representations are more often than not task specific: there is no single universal set of features that solves all the problems in computer vision. Consequently, feature learning for computer vision tasks is not a problem, rather a set of problems, a full fledged field of research per se.

In this thesis, we seek to learn good, semantically meaningful representations for some of the popular computer vision tasks, such as visual classification, action recognition and so on. We study and employ a variety of existing feature learning approaches, and devise novel strategies for learning representations on these tasks. Additionally we compare our methods with the traditional approaches. We discuss the design choices we make, and the effects of varying the parametric variables in our approaches. We provide empirical evidence to show our representations are better at solving the tasks at hand than the traditional ones.

To solve the task of action recognition, we devise a novel PLS kernel that employs Partial Least Squares (PLS) regression to derive a scalar measure of similarity between two video sequences. We use this similarity kernel to solve the tasks of hand gesture recognition and action classification. We demonstrate that our approach significantly outperforms the state of the art approaches on two popular datasets: Cambridge hand gesture dataset and UCF sports action dataset.

We use a variety of approaches to tackle the popular task of visual classification. We describe a novel hierarchical feature learning strategy that uses low level Bag of Words visual words to create "higher level" features by making use of the spatial context in images. Our model uses a novel *Naive Bayes Clustering* algorithm to convert a 2-D symbolic image at one level to a 2-D symbolic image at the next level with richer features. On two popular datasets, Pascal VOC 2007 and Caltech 101, we demonstrate the superiority of our representations to the traditional BoW and deep learning representations.

Driven by the hypothesis that most data, such as images, lies in multiple non-linear manifolds, we propose a novel non-linear subspace clustering framework that uses K Restricted Boltzmann Machines

(K-RBMs) to learn non-linear manifolds in the raw image space. We solve the coupled problem of finding the right non-linear manifolds in the input space and associating image patches with those manifolds in an iterative Expection Maximization (EM) like algorithm to minimize the overall reconstruction error. Our clustering framework is comparable to the state of the art clustering approaches on a variety of synthetic and real datasets. We further employ K-RBMs for feature learning from raw images. Extensive empirical results over several popular image classification datasets show that such a framework outperforms the traditional feature representations such as the SIFT based Bag-of-Words (BoW) and convolutional deep belief networks.

This thesis is an account of our efforts to do our bit to contribute to this fascinating field. We admit that research in this field will continue for a long time, for solving computer vision is still a distant dream. We hope that we have earned the right to say one day, in retrospect, that we were on the right track.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Human beings use their eyes and brains to sense and understand the world. Computer vision is the discipline that intends to produce machines that can evolve similar behaviour. The vision community has always wondered what it is that makes our human vision so special. Research into how the human brain functions has given us useful clues, and it is now speculated that the way we represent/organize the world is as crucial as how we use these representations to learn and understand. Many attempts have been made over the years to develop rich, semantically meaningful models that represent the world. Nevertheless, there is no clear consensus on how to portray the world. Our prior experiences have only revealed that good representations are task specific: there are no universal features that solve all the vision tasks. In this chapter, we introduce features, and delve into the subject of what it means for a machine to have understood the world.

1.1 Features

Typical images / videos are huge in terms of the number of pixels. Even a small, passport sized, photograph in a digital format typically has several hundred thousand pixels. While most modern computers today can process such high dimensional data, working with raw images is undesirable because of (a) irrelevant, redundant information, (b) no built-in invariance to external parameters (illumination changes, orientation changes, scaling, translation and other distortions), and even (c) the lack of organization. Instead, we seek good, semantically meaningful representations of the data. We refer to these representations as "features" and the process of computing these features is called *feature extraction*. Feature extraction is a special form of dimensionality reduction [1].[1] Good features are characterized by several qualities: (a) the representation size is manageable by the learning method (the speed and memory requirements are fulfilled), (b) the representation captures all / most of the relevant information, (c) all / most of the redundancy in data is eliminated, and (d) invariance to external parameters such as illumination changes, orientation changes, scaling, translation and other distortions is maximal. Which

[1] While not all features are smaller in size than the raw data, this property holds true in most computer vision tasks.

information is relevant and which is not is dictated by the task at hand. Consequently, good features are task specific. This can be regarded as the *No Free Lunch* theorem in the representation domain.

1.2 Computer Vision Tasks

The holy grail of computer vision is to build machines that can *see* in the manner humans do. After being exposed to millions of images/videos, the hope is that such a machine vision system would be able to understand an image/video it has never seen before. *Understanding an image* is a vague phrase and can mean different things in different contexts. Before we can delve into what exactly we mean by understanding an image, we should fix the context. In this section, we talk about a variety of computer vision tasks and discuss what it means to have understood an image with reference to these contexts.

Computer vision, as a scientific discipline, aims to develop machines that extract information from images. The image data may take many forms, such as video sequences, views from multiple cameras, or high-dimensional data from a medical scanner. This information is then applied to making decisions.

Computer vision has been applied to making all sorts of decisions in the industry. Applications can be as simple as counting automobile parts speeding by on a production line or as complex as robots looking at and understanding the world around them. As a matter of fact, most fields that require capturing and processing image data use computer vision as their core. Other applications of computer vision include (a) navigation devices used in vehicles or robots, (b) visual surveillance, (c) organizing databases of images or videos, (d) human computer interaction devices, (e) medical image analysis, (f) modeling terrains and topographies, (g) automatic inspection in manufacturing applications and so on. All these applications employ a range of computer vision tasks which can be solved using a variety of methods.

Some of the popular computer vision tasks are scene reconstruction, content based image retrieval, image segmentation, object detection, image classification, instance recognition, indexing, clustering, image restoration, pose estimation, event detection, object tracking, motion estimation, action recognition and so on. In this thesis, I have worked specifically on few of these tasks, (a) clustering, (b) action recognition and (c) image classification. In the rest of this section, I will describe each of these tasks in detail.

1.2.1 Clustering

Clustering is one of the most popular and perhaps one of the most important *unsupervised* learning problems. Clustering seeks to find structure in data and is driven by the hypothesis that data is not randomly distributed across the feature space but has inherent high density regions with few outliers and/or background noise points.

Clustering could be defined as "the process of organizing objects into groups whose members are similar in some way". A cluster is therefore a collection of objects which are "similar" to each other and

are "dissimilar" to the objects belonging to other clusters. Figure 1.1 describes a clustering of points in the 2D space where we define similarity between points in terms of the Euclidean distance between them. In other words, points close to each other are *clustered* together.

Clustering seeks to find inherent high density regions in the data distribution. In this context, understanding refers to gaining insights into the nature of data which equips us with means to identify these high density regions in the space (clusters).

In computer vision, clustering has been treated both as a task and the means to solve a task. More specifically, clustering has been used for the following:

- directly clustering features/data such as images, videos. Applications include data organization, data analysis, segmentation, and so on. In chapter 5, we employ clustering to organize the MNIST dataset.
- unsupervised discovery of sub-categories, which helps relax the huge intra-class variation in data from the same category. This has been applied in visual recognition tasks [68], among others.
- learning representations such as *Bag of Words* models. In chapter 4, we devise a novel *Naive Bayes* clustering approach to learn hierarchical *Bag of Words* representations.

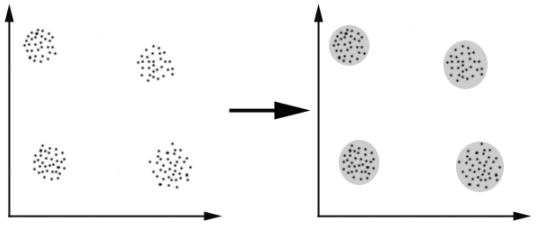


Figure 1.1: Toy example: Clustering of points in 2D space. The similarity between points is inversely proportional to the Euclidean distance between them.

1.2.2 Action Recognition

Action recognition is another popular computer vision task. Action recognition refers to the task of analyzing a video sequence and identifying the activity going on. This includes hand gesture recognition and human activity recognition, among other tasks. Hand gesture recognition refers to the task of mapping a sequence of (hand) images denoting a gesture to an event (or the gesture). This has applications in human computer interation, sign language interpretation and so on.

Human activity recognition is another important area of computer vision research and applications that falls under the broad category of action recognition tasks. Figure 1.2 shows some human activities.

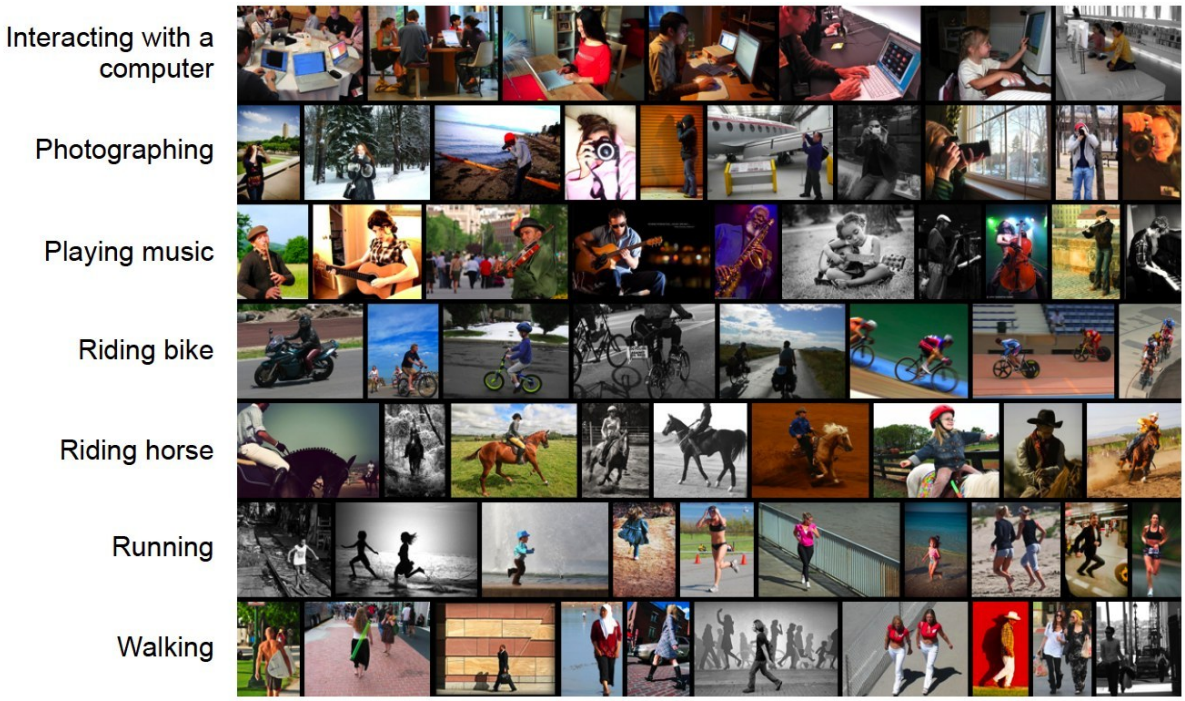


Figure 1.2: Human Actions, represented as a sets of images for convenience here, are intended to be seen as a set of videos/image sequences. Picture courtesy: V. Delaitre, I. Laptev and J. Sivic

Human activity recognition involves analyzing a video sequence and predicting the activity in the video. Recognition of human actions can be seen as video interpretation. While a few human actions such as reading, playing a guitar may be inferred from still images, some (for instance sitting down, standing up) can only be inferred using temporal context.

Understanding in the context of activity recognition can be understood as interpretation of ongoing events and their context from video data. Applications include surveillance systems, patient monitoring systems, and a variety of systems that involve interactions between persons and electronic devices such as human-computer interfaces. Action recognition is a challenging problem because videos contain not only spatial but also temporal information, which are usually hard to capture simultaneously by the traditional image representations. We tackle action recognition in chapter 3.

1.2.3 Visual Classification, Object Recognition

Visual (image) classification, a classical problem in computer vision, image processing, and machine vision, involves determining whether or not the image data contains some specific object, feature, or activity. In other terms, visual classification refers to the task of classifying an image according to its visual content, for example whether it contains an aeroplane or not. In most visual classification settings, we are given a set of candidate classes / categories (such as mountain, aeroplane, cityscape, and so on)

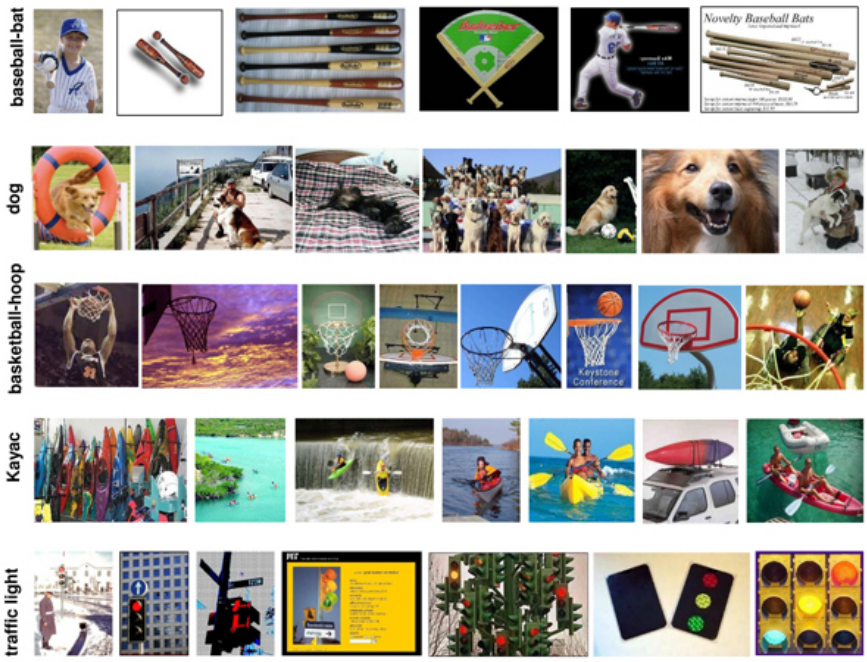


Figure 1.3: Image classes (Caltech 256). In most images in Caltech, the presence of an object of interest in the image determines the class label. Picture courtesy: Anna Bosch and Andrew Zisserman

and an unseen test image and our goal is to predict the most appropriate class label for it. Figure 1.3 shows sample images for a few candidate classes in the dataset Caltech 256. A visual classification task can be either scene classification or object recognition depending on the set of candidate classes: it is scene classification in case all the classes are scenes (night-time, cityscape, under-water) or object recognition in case all the classes are objects (aeroplane, bicycle, cow). A related problem, fine grained scene classification, refers to visual classification in a setting where the classes are similar in appearance (bedroom, kitchen, office and so on).

Content-based image retrieval is a popular application of visual classification. It refers to finding all images in a larger set of images which have a specific content. The content can be specified in many ways, for example in terms of queries like “show me all images similar to image X”, or “show me all images which contain many houses, are taken during winter, and have no cars in them”, and so on.

In a visual classification setting, understanding an image would mean being able to predict the class of an image or a video. We describe our approaches to visual classification in chapters 4 and 5.

Chapter 2

Representations in Computer Vision

In this chapter, we talk about the traditional representation models used ubiquitously in computer vision tasks. We start with a discourse on popular features and feature learning methods in computer vision literature, describe an example of a feature learning model called the Restricted Boltzmann Machine, and the rest of the chapter describes the two prominent orthogonal schools of thought in the feature learning domain.

2.1 Features

As described in chapter 1, features are means of representing the world (images / videos) in computer vision and feature extraction is the process of computing these features. Typical images / videos are huge in terms of the number of pixels, and feature extraction is a special form of dimensionality reduction of the image data. A good feature is (a) compact so the representation size is manageable by the learning method, (b) captures most of the relevant information in the data, (c) eliminates most of the redundancy in data, and (d) is invariant to external parameters such as illumination changes, orientation changes, scaling, translation and other distortions.

Feature learning is a form of dimensionality reduction. Dimensionality reduction can be seen as subspace projection, where we compute features by expressing them as linear combinations of the data variables, while still describing the data with sufficient accuracy. Subspace projection can either be linear or non-linear depending on the nature of subspace we use for projection of the data into the feature space. Consequently, we can have both linear and non-linear feature representations. The choice of the dimensionality reduction method rests with experts who have domain knowledge. However, in the absence of expert knowledge, general dimensionality reduction methods come in handy. Some of the popular dimensionality reduction methods are, (a) Principal component analysis, (b) Semidefinite embedding, (c) Multifactor dimensionality reduction, (d) Multilinear subspace learning, (e) Nonlinear dimensionality reduction, (f) Isomap, (g) Kernel PCA, (h) Multilinear PCA, (i) Latent semantic analysis, (j) Partial least squares, (k) Independent component analysis.

In the recent years, much research has gone into designing good features for images. Feature learning for computer vision tasks has evolved into a full fledged discipline. These new age features can be very broadly classified into two categories:*hand crafted* and *learnt*. These categories also represent the two orthogonal directions of research in feature learning: the former relies on domain knowledge to craft representations by hand and the latter on the belief that richer representations can be "learnt automatically" from the data using hierarchical models.

Hand crafted feature extraction relies on image processing methods to detect artefacts in the image such as (a) edges, (b) corners, (c) blobs, (d) ridges, (e) curvature, (f) predefined templates, (g) known shapes such as lines, circles, ellipses and so on, (h) parametric shapes, (i) active contours and so on. The final feature representation is often a histogram of the distribution of these artefacts. Thus, hand crafted features summarize the image data in terms of the distribution of the occurence of the artefacts they seek to discover. SIFT [43], SURF[6], GIST [49], and HOG [15] are popular hand crafted features. SIFT detects and counts artefacts by computing difference of Gaussians around points of interest, SURF uses Haar wavelet responses around interest points to detect Hessian blobs. HOG detects and counts simple gradients, and GIST uses gabor filters to detect artefacts.

Learnt representations employ deep learning architectures modeled after the human brain to capture rich, semantically meaningful representations. This feature learning community believes the human brain has a deep structure (a huge network of neurons), and humans organize their ideas hierarchically, through composition of simpler ideas. Like hand crafted approaches, deep learning approaches also seek to discover simple artefacts in the data. However, unlike the hand crafted feature extraction approaches, these artefacts are learnt (are free to take any form) by looking at the data and not detected by matching premeditated templates. Besides, deep architectures assemble the learnt simpler artefacts to learn bigger features: pixels are assembled into edges, edges into object parts, and object parts into objects (figure 2.4).

Features of an image can also be categorized on the basis of whether or not they represent the whole image. For instance, while GIST features represent the entire image, SIFT features are local and computed around interest points in an image. Consequently, while the GIST feature is always a fixed sized representation, SIFT representation of an image can vary in size, depending upon the number of interest points in the image. We talk about the various interest point detection strategies in the next section. Most popular scene or object recognition models first compute low-level (local) features such as SIFT descriptors over interest points in an image and then transform / pool them into global representations such as bag of words models. Thus, the current trend in feature representation [34] dictates a two step approach to image representation: (1) a coding step, which involves computing local descriptors suited to the task, and (2) a pooling step, which summarizes the coded features over larger neighborhoods, often the entire image. We describe examples of this approach in detail in the following sections.

2.1.1 An example of a feature learning method: Restricted Boltzmann Machines

Restricted Boltzmann Machines (RBMs), first introduced by Smolensky in [66] are undirected, energy-based graphical models that learn a non-linear subspace by minimizing reconstruction error. The input data can then be projected onto the learnt subspace; RBMs can thus be seen as non-linear dimensionality reduction tools or simply put, feature extractors.

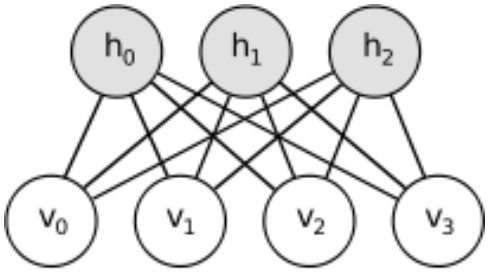


Figure 2.1: A simple Restricted Boltzmann Machine

RBMs are two layered, fully connected networks that have a layer of input/visible variables and a layer of hidden random variables. They model a distribution over visible variables by introducing a set of stochastic features. In most computer vision applications, the visible units correspond to the pixel values and the hidden units correspond to visual features. A learnt RBM represents a subspace in the pixel space, and can be used to project the pixels (image) to the feature space. In chapter 5, we describe an approach to feature learning using RBMs. The rest of this section describes the mathematical model of an RBM.

For an RBM with I visible units $v_i, i = 1, \ldots, I$ ($v_0 = 1$ is the bias terms), J hidden units $h_j, j = 1, \ldots, J$ ($h_0 = 1$ is the bias term) and symmetric weighted connections between the visible and hidden layers denoted by $\mathbf{w} \in \mathbb{R}^{(I+1)\times(J+1)}$ (these include asymmetric forward and backward bias terms), the activation probabilities of units in one layer are computed based on the states of the opposite layer:

$$Pr(h_j|\mathbf{v}) = \sigma\left(\sum_{i=0}^{I} \mathbf{w}_{ij} v_i\right) \tag{2.1}$$

$$Pr(v_i|\mathbf{h}) = \sigma\left(\sum_{j=0}^{J} \mathbf{w}_{ij} h_j\right) \tag{2.2}$$

$\sigma(\cdot)$ is the sigmoid activation function. The RBM energy function, defined as the negative log probability of a configuration of states $(\mathbf{v}, \mathbf{h})$ is given by:

$$-logPr(\mathbf{v}, \mathbf{h}) = E(v, h) = \sum_{i,j} v_i h_j w_{ij} \tag{2.3}$$

Training the RBM thus involves learning the RBM weights and biases that minimize this energy $E(\mathbf{v}, \mathbf{h})$. Ideally the RBM parameters would be learnt by maximizing the likelihood. This objective function is

called the *alternative Gibbs sampling*. However, computing this maximum likelihood involves an exponential number of terms, which makes the training slow and unmanageable. Hinton, in [25], proposed an objective function called *contrastive divergence* (CD) which is an approximation to the maximum likelihood objective function and can be efficiently minimized. In the CD-1 forward pass (visible to hidden), we activate the hidden units h_j^+ from visible (input) unit activations v_i^+ (Eq.5.1). In the backward pass (hidden to visible), we recompute visible unit activations v_i^- from h_j^+ (Eq.5.2). Finally we compute the hidden unit activations h_j^- again from v_i^-. The weights are updated using the following rule:

$$\Delta w_{ij} = \eta(< v_i^+ h_j^+ > - < v_i^- h_j^- >) \tag{2.4}$$

where η is the learning rate and $<\cdot>$ is defined as the mean over N examples. The reconstruction error for any sample is computed as:

$$\epsilon = \sum_{i=1}^{I} (v_i^+ - v_i^-)^2 \tag{2.5}$$

The power of RBMs can be attributed to the fact that they are a generic framework for learning non-linear subspaces, make no assumptions about the sub-spaces, use a standard energy based learning algorithm, and can model subspaces of any degree of complexity via the number of hidden units making them most suitable as general purpose sub-space learning machines.

2.2 Bag of Words Model for Image Representation

The bag-of-words representation for images was introduced in [65, 14], inspired by the success of the bag-of-words (BoW) features in document classification. The text based BoW models typically represent documents as histograms of words in the dataset and ignore the order of words. Similarities between two documents are then approximated by computing similarities between their histograms (BoW representations).

BoW models for images are based on the same hypothesis: images are composed of smaller artefacts (visual words) and counting the number of different types of visual words in the image gives a reasonably accurate description of the image. For instance, consider the task of recognizing images of cars. Cars come in a variety of shapes and sizes: vintage, sports, convertibles, sedans, SUVs, wagons, hybrid and so on. A good representation should thus be invariant to the variation in shapes and sizes. One way to achieve this objective is to not look at the car itself, but to recognize smaller objects (artefacts)[1] in the car: the wheels, the steering wheel, the rearview mirrors, the doors, the head lights, the tail lights, bumpers and so on. Counting these smaller artefacts in the image gives us a fair idea of whether the image has a car or not. Figure 2.2 describes a BoW representation of a female torso. The good thing about this representation is the fact that we didn't impose any restrictions on the size, the position and the orientation of the car in the image. In the traditional BoW model, we completely disregard the spatial

[1]The artefacts we name here are merely for simplicity. In most BoW representations, the actual artefacts we come across are much smaller in size and are relief edges and patterns.

context of the image. Thus, BoW models are inherently invariant towards changes in scale, position and orientation. Having said this, we must add that ignoring the spatial context can be a blessing, or a curse, depending on the task at hand. In the rest of this section, we discuss the classification pipeline, the implementation details and issues associated with traditional BoW models in computer vision.

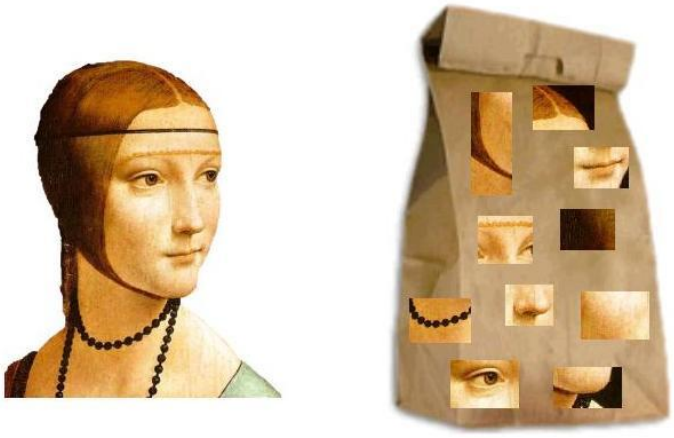


Figure 2.2: Bag of Words. The image shows an object (a torso of a woman), and what its traditional BoW representation would look like.

The traditional BoW classification pipeline involves the following steps:

- Computing local features on interest points in the image.
- Vector Quantization.
- Image histogram computation.

We now describe each of these steps.

2.2.1 Features

Computing the BoW model for images involves first computing low level features such as SIFT [43], SURF [6], HOG [15] over interest points in the images. Interest points are points in images around which we compute features. Classically, interest points denoted points in the image for which the signal changed two-dimensionally. There are various strategies in literature for choosing these interest points. Interest points could be chosen (a) by an interest point detection algorithm as in [64, 14], or (b) by densely sampling the image in a grid as in [18, 76], (c) segmentation based sampling [5] or (d) even randomly [70]. Densely sampled interest points have been shown to work well on a variety of computer vision tasks [18], and are the most prevalent. The local features computed over the interest points are referred to as keypoints.

SIFT features have enjoyed success on a variety of visual classification tasks. SIFT features (a) work in a normalized intensity space, so are partially invariant to changes in illumination, and (b) are invariant to uniform scaling, orientation, and partially invariant to affine distortion by design. Most recent BoW representations have employed dense SIFT features which are SIFT features computed over interest points in a dense grid.

2.2.2 Vector quantization

These keypoints / local features in the image are vectors of real values. However, a BoW representation is a histogram of visual words. Thus, we need some sort of vector quantization method that converts these real valued keypoints to symbolic visual words. Usually, a simple clustering method such as K-Means or agglomerative clustering is employed for vector quantization. The most popular choice is K-means. We use K-Means on the keypoints to get K cluster centers; we call each cluster center (also called the centroid or cluster representative) a visual word. The set of K cluster centers is called the vocabulary / dictionary / codebook of the dataset. Vector quantization involves replacing / approximating each keypoint with its nearest visual word. Vector quantization allows us to convert a real valued image to a symbol image (each local feature is now seen as a symbol / visual word). Subsequent processing is done on the symbol image.

Vector quantization imparts BoW representations with several desirable attributes that make them suitable for a variety of computer vision tasks. BoW models are efficient, reasonably discriminative and robust to a number of nuisance parameters such as scale, translations, illumination changes, and other deformations. This discretization of SIFT or other features to a symbol space is necessary in large scale problems because (a) computing distances between a huge number of descriptors is inefficient, and (b) storing all the descriptors is not feasible, instead we store only the symbol (visual word) identifier per descriptor. After this vector quantization, the Euclidean distance between the high dimensional features (128 in case of SIFT) is approximated by a $0 - \infty$ metric. It is assumed that features corresponding to the same visual words are absolutely identical, while features corresponding to different visual words are totally different. This computational convenience comes at the cost of discriminative power of the representation.

2.2.3 Image histogram computation

Computing the BoW model of an image requires building a histogram of the visual words in the image. This is done by simply counting the occurences of each visual word in the symbol image.

Histogram binning adds another dimension to BoW models. It is noteworthy that the traditional BoW representation is always a fixed $K-$dimensional representation, where K is the vocabulary size, despite the difference in image sizes/number of interest points per image. This is particularly suited for most of the learning algorithms that assume that the input space has a fixed size. To make the representation invariant to these parameters such as number of interest points, it is customary to normalize the

histogram. The most common means of normalization is the $L1$ normalization where we simply divide the representation by the sum of the histogram elements, so that the final representation adds to 1. A BoW histogram thus summarizes the image as a distribution of the low level visual words in the image. This distribution is also referred to as term-frequency in some contexts.

2.2.4 Beyond BoW: Spatial Pyramids

As described in the preceding sections, the traditional BoW model disregards the spatial context in the image completely. While on one hand, this imparts the representation invariance to distortions, traslations, orientation and scaling, it also discards discriminative information that is readily available. For instance, in recognizing faces, the relative positions of the eyes, nose, lips, ears and so on can be very informative and discriminative. Over the years, much research has gone into enhancing BoW representations by incorporating spatial context. One of the most noted works in this direction is spatial pyramids [37].

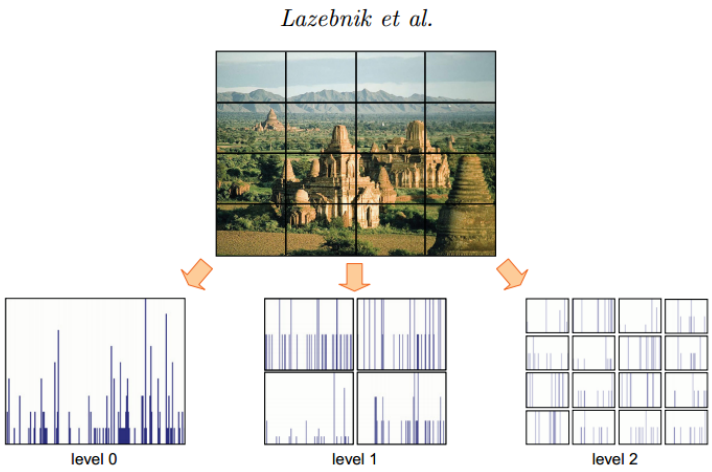


Figure 2.3: Spatial Pyramid Matching. Picture Courtesy: [37]

This approach computes BoW representations for image regions at different locations in various scales. In other terms, we compute features hierarchically, dividing the image into a grid like structure at each level and computing a BoW representation for each cell in the grid. The final image representation is the concatenation of these smaller BoW representations.

Figure 2.3 demonstrates this approach. At level 1, we compute a BoW representation for the entire image. This is denoted by the first histogram in the figure. At level 2, we divide the image into 4 symmetric regions and compute a BoW representation for each of these 4 parts. This is denoted by the second histogram. At level 3, we divide the histogram symmetrically into 16 regions and the BoW representations for each of these are denoted as the third histogram. We can further partition the image into smaller regions if we feel the need to do so. The final image representation is the concatenation

of all these histograms. It should be noted that the representative power of spatial pyramids comes at a price: the size of the representation increases manyfold per level. In figure 2.3, if the codebook size is K, the level 1 pyramid has a representation size of K, the level 2 pyramid has a representation size of $4K$ and the level 3 pyramid has a representation size of $16K$. The final representation will thus have a size of $21K$. It goes without saying that these grids need not be symmetric. As a matter of fact, most recent works experiment with the grid structure. For instance [29] uses 8 spatial regions for the Pascal VOC 2007 dataset, by dividing the image in 1×1 (full image), 3×1 (three horizontal stripes), and 2×2 (four quadrants) grids. It should be noted that each spatial region is normalized individually prior to stacking.

2.2.5 Practical Issues

Learning BoW representations for vision tasks involves making few design choices, like how to generate interest points and which local features to choose. As already stated, dense SIFT features work well for a variety of tasks. Choosing the vocabulary size is another crucial decision. It has a direct bearing on the representation size and the representation power. Traditionally, the BoW representation size is proportional to the codebook size. If the vocabulary is too small, the visual words are not representative of all patches, while if the vocabulary is too large, quantization artefacts and overfitting occur. The size of the codebook doesn't have a closed form solution and is decided empirically.

Another issue while computing BoW representations is scalability. Conventionally, we use a subset of the training features to compute the vocabulary. Thus, the local feature descriptors are sampled from the training set and K-means is employed to cluster this subset. People use different sampling strategies for this task: (a) randomly sampling the desired number of descriptors from the entire dataset, (b) sample a fixed number of descriptors from each image, (c) sample a fixed number of descriptors from each class, and so on.

Apart from the above parameters, proper normalization of the representation is also necessary. $L1$ and $L2$ normalizations are popular in literature. In popular computer vision tasks such as visual classification and recognition, the choice of the classifier is also crucial. SVM classifiers have been used in most recent works; although neural networks and nearest neighbour classifiers have also been described in literature. SVM learning is intractable in very high dimensional spaces, so fixing the representation size becomes all the more crucial. Many attempts in this field have attempted development of distance functions and kernel functions that enhance the accuracy of the predictions.

2.2.6 A Note on Visual Vocabularies

Visual vocabularies allow efficient indexing for local image features. There are several methods of codebook construction in literature. The most common, and most popular method of learning a visual vocabulary or a codebook is clustering. Typically, the vocabulary is constructed by clustering the local image features using the K-Means algorithm [37, 29]. K-Means partitions the space well, minimizing

the variance between the data points and the cluster centers. However, findings in [27, 10, 54] suggest that the most frequently occuring words are not necessarily the most discriminative. Motivated by this hypothesis, a number of methods have attempted to build discriminative vocabularies by (a) alternative clustering algorithms [27, 42], (b) using class label information [47, 80, 83], and (c) even creating class-specific categories as in [51, 35, 69].

Besides the popular clustering approaches, a vocabulary may be constructed by a manual labeling of image patches with semantic labels, as in [73, 77]. These semantic vocabulary approaches are more intuitive because they try to express the image in terms of what the constituent visual words in it mean.

Vocabulary trees [48], where we use hierarchical K-Means by choosing a branching factor and number of levels, allow recursive division of the feature space. These enable us to efficiently reduce the computational cost of assigning visual words to features, from linear to logarithmic in the size of the vocabulary. This allows us use much larger codebooks. Also, experimentally, it was shown that these visual words are more specific and are particularly suited to matching specific instances of objects.

It is worth considering what it is that a visual word captures. This depends on a variety of factors: (a) what features were used to construct the vocabulary, (b) the size of the codebook, (c) the vector-quantization strategy used, and (d) how the interest points in the original images were chosen. Despite all these parameters, it is reasonable to assume that patches assigned to the same visual words have a similar appearance in the low-level feature space. Also, in most practical solutions (where we use unsupervised methods for codebook construction), there is usually no guarentee of correlation between the visual words and the object parts/artefacts.

One of the major design decisions when learning the visual codebook is choosing the data used to construct it. The most common protocol is to use a subset of the features from the training set to build the vocabulary. Also, in most cases, the training set is compiled to contain samples similar (often coming from the same data source) to the ones that would be present in the testing set. This often leads to the most accurate results. Also, when we intend to learn a vocabulary for a set of categories, it is customary to sample descriptors from training samples covering all categories, ensuring that all categories are represented.

Choosing the feature extraction method and the interest point strategy are also crucial to the types of visual words generated. These in turn affect that similarity measured between the visual word distributions. Domain knowledge comes in handy at this point. Recent approaches have shown that a dense grid of interest points for collecting local features works well on a recognition tasks. Usually, we collect dense features at multiple scales [29]. Dense features are more suited to the task of category recognition while features computed at interest points are usually preferred for recognizing instance of objects.

Thus there are many choices available when creating a visual vocabulary. Each of these choices affects the semantics of our representation and the accuracy of our solution. However, there is still no clear consensus on the *correct* means of learning a visual vocabulary. The analogies between textual and visual data are only approximate: textual words are discrete, symbolic language defined semantic con-

structs, while natural images are continuous, complex entities. “Real sentences have a one-dimensional structure, while images are 2D projections of the 3D world.” [24].

2.3 Deep learning

Although representations based on hand crafted features like SIFT and HOG have been reasonably successful on many computer vision tasks, they are not the only means to our end. One of the major role players behind the success of hand crafted features are the subsequent components in the model pipeline. Use of complex models such as SVM classifiers, distance functions and kernels of different kinds have all contributed generously to the success of methods based on hand crafted features. In a manner of speaking, complex models are used to compensate for the simplicity of these representations. An alternative approach is to continue to “enrich” low level features by using hierarchical feature learning paradigms such that after several levels of semantic enrichment, the features are meaningful enough that even simple models can accomplish complex tasks.

A school of researchers believes that good internal representations of the data are learnt from the data and are hardly hand crafted at all. The primary issues with hand-crafted features are (a) crafting features from hand requires domain knowledge, (b) the features do not generalize well to other domains, and (c) only capture low-level edge information. Many recent attempts in feature learning have intended to design features that effectively capture mid-level cues (e.g. edge intersections) or high-level representation (e.g. object parts). Driven by the idea that computer vision is more than just pictures, over the years, many researchers have asked themselves the same question, ”can we *automatically* learn a good feature representation?” Figure 2.5 shows the different kinds of data that we encounter while working on computer vision problems. Recent developments in machine learning have shown how hierarchies of features can be learned in an unsupervised manner directly from data. These endeavours have resulted in *Deep Learning* architectures.

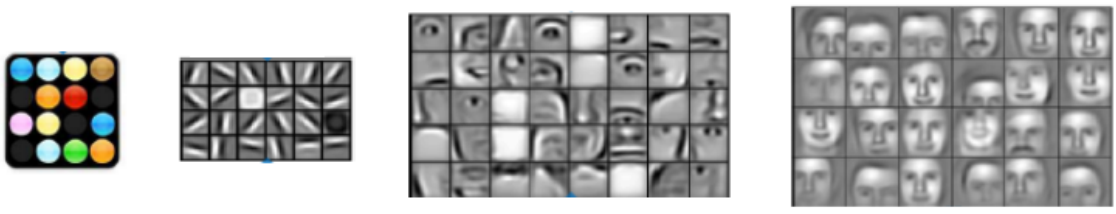


Figure 2.4: Feature Hierarchies. From left to right, (a) Pixels, (b) Edges, (c) Object parts, and (d) Objects.

Deep learning networks [26, 40] and convolutional networks [38] are driven by the idea that good internal representations are hierarchical and can be learned directly from the data. These networks have

multiple hierarchical layers (also called *feature maps*) stacked together; each feature map learns artifacts in the image by assembling smaller artifacts learnt by the preceding feature maps. As described in figure 2.4, pixels are assembled into edges, edges into object parts, object parts into objects, and objects into a scene; deep learning thus exploits the spatial information in the images. These levels represent the feature hierarchy.

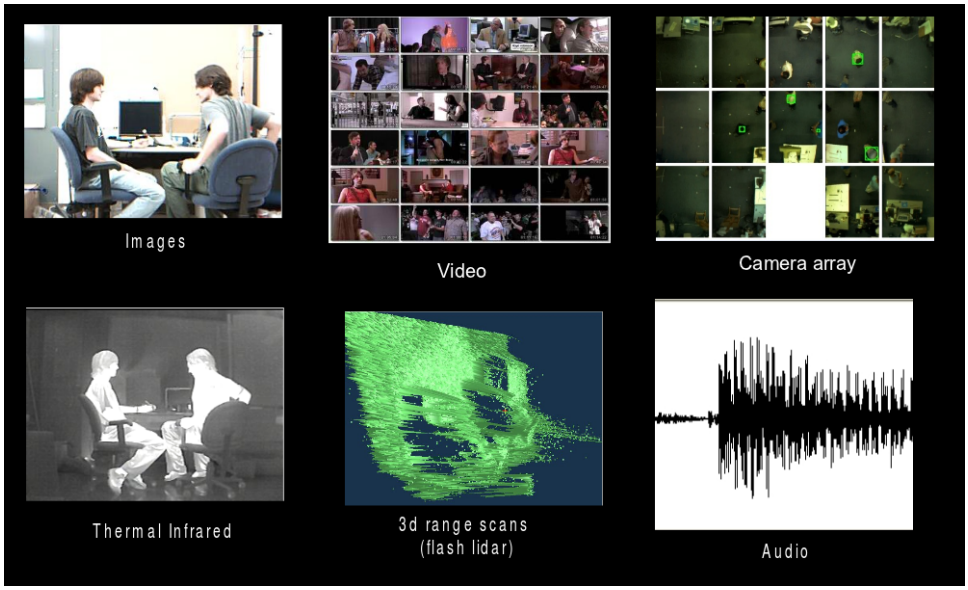


Figure 2.5: Different types of data encountered in computer vision. Picture Courtesy: Andrew NG

Deep learning architectures, often referred to as *self taught learners*, draw inspiration from the human brain which is known to learn using its network of neurons. Consequently most deep learning models are based on neural networks. Traditionally, deep learning is performed in an unsupervised manner, which relaxes the constraint that data needs to be labeled. This allows us to learn from the abundant unlabeled data. Convolutional neural networks, deep belief networks and convolutional deep belief networks are some popular examples of deep learning methods. In the next section we describe convolutional deep belief networks (CNNs) because they best demonstrate the characteristics of a deep learner.

2.3.1 An example of deep learning models: Convolutional Neural Network

CNNs are a special type of neural networks, also trained using the back propagation algorithm. They differ from traditional neural networks in their architecture. Like most deep learning architectures, they are designed to recognize visual patterns directly from pixel images with minimal preprocessing. CNNs recognize patterns with extreme variability and are robust to distortions and simple transformations. In figure 2.6, the CNN (feature extractor) is coupled with a multilayer neural network which acts as a classifier. This is in line with our remark that deep learning methods *enrich low level features by*

using hierarchical feature learning paradigms such that after several levels of semantic enrichment, the features are meaningful enough that even simple classifiers do the job. Thus, CNN relies as much as possible on learning in the feature extractor itself. Figure 2.6 describes the structure of a convolutional neural network (CNN).

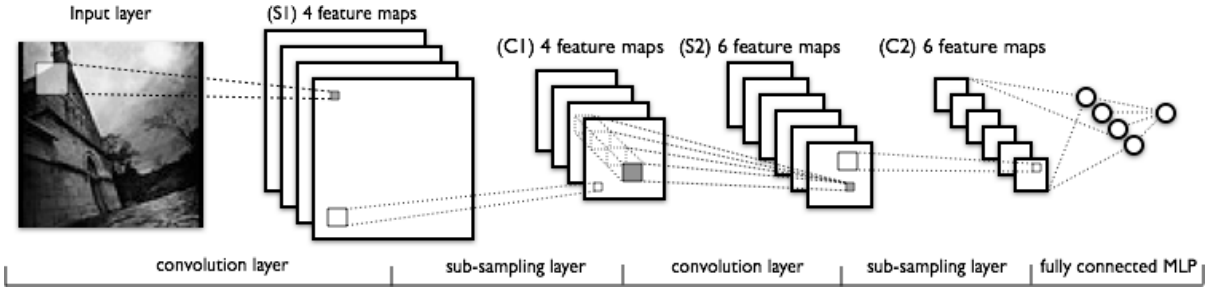


Figure 2.6: Convolutional Neural Network

CNNs combine three architectural strategies to ensure some degree of shift, scale and distortion invariance:

- local receptive fields
- shared weights (weight replication)
- spatial or temporal sub sampling

Local receptive fields: The input layer (or a plane) in a CNN receives raw images that are approximately size normalized and centered. Each unit in a layer receives inputs from a set of units located in a small neighbourhood of the previous plane (local receptive fields). With local receptive fields, neurons extract elementary visual features like oriented edges, end points, corners etc. These features are then combined by subsequent layers in order to detect higher order features.

Shared weights: Distortions or shifts of the input can cause the position of salient features to vary. Elementary feature detectors that are useful on one part of the image are likely to be useful across the entire image. This knowledge can be applied by forcing a set of units whose receptive fields are located at different positions in the image to have identical weight vectors. Units in a layer are so organized in planes within which all the units share the same set of weights. The set of outputs of the units in such a plane is called a feature map. In figure 2.6, units in a feature map perform the same operation on different parts of the image. Typically, we have several feature maps for the same location so we compute multiple features. Let us suppose a unit in a feature map receives input from a 5x5 patch (called the receptive field) in the input field, hence has 25 inputs. The receptive fields of horizontally contiguous units overlap by 4 columns and 5 rows. In figure 2.6, at each input location 4 different types of features are extracted by four units in identical locations in the 4 feature maps (this is denoted by the feature maps $S1$). A sequential implementation of a feature map would scan the input image with

a single unit that has a local receptive field and store the states of this unit at corresponding locations in the feature map. This operation is equivalent to a convolution, followed by an additive bias and an activation function, hence the name convolutional network.

Spatial sub sampling: If the input image is shifted, the same feature is captured by a different unit with a different receptive field. This is the basis of robustness of convolutional networks to shifts and distortions. Once a feature has been detected, its exact location becomes less important (and harmful). Only its approximate position relative to other features is relevant. A simple way to reduce the precision with which the position of distinctive features are encoded in a feature map is to reduce the spatial resolution of the feature map.

Sub-Sampling layers perform local averaging and a sub-sampling reducing the spatial resolution of the feature maps, hence reducing the sensitivity of the output to shifts and variations. In figure 2.6, the 2^{nd} hidden layer ($C1$) is a sub-sampling layer. Since there are 4 feature maps in the 1^{st} hidden layer ($S1$), there are 4 feature maps in this sub-sampling layer too. The receptive field is a 2x2 patch in the previous layer. Each unit here performs averaging, followed by multiplication with a trainable coefficient, addition of a trainable bias and passing through a sigmoid function. Contiguous units have non-overlapping receptive fields, hence half the number rows and columns with respect to previous layer.

It is to be noted that CNNs are feature extractors per se. In figure 2.6, the feature extractor is coupled with a neural network. Thus, while the feature extraction module is totally unsupervised, we need class labels to train the classifier. This is in line with our earlier remark that traditional deep learning architectures perform unsupervised feature learning.

2.3.2 Discussion

Traditional deep learning architectures are hierarchical feature learners. These deep models typically work on *overlapping* patches at each level (the overlap takes care of small translations) and summarize the features learnt in a neighbourhood by a pooling method. In case of a CNN, we described an average pooling method where the features computed over a region (called a cell) are averaged to give the feature representation of the cell. There is a second form of pooling called max-pooling, where the feature representation of the cell is the maximum of the features in the cell. The choice of pooling strategy is a crucial one. While in a generative framework, average pooling might seem to be the better option, in a discriminative classification setting, max-pooling is generally preferred. Convolutional networks alternate between feature maps and pooling layers to achieve invariance to small translations and distortions.

Deep learning models learn richer, semantically meaningful representations. However, this richness comes at a cost. These models are complex in terms of their training and architecture. Designing a deep learner to accomplish a task involves making many policy desicions: number of layers, number of units per layer, choosing activation functions, choosing pooling strategies and so on. Training a deep learner usually requires us to learn many parameters iteratively in a gradient descent procedude, and as such

is slow. Besides, other issues associated with gradient descent methods such as deciding the objective function, choosing the learning rate so as to avoid local optimal states, choosing momentum, preventing over fitting, and so on futher make the training sensitive to a lot training variables.

Deeply learnt representations have enjoyed immense success on tasks such hand written and machine printed character recognition. One of the primary reasons for the success of deeply learnt architectures is the fact that they exploit almost all the spatial information in the image. As shown in figure 2.4, deep architectures learn objects in the image by assembling the already learnt smaller artifacts. While spatial context may seem like a very desirable characteristic of an image, relying too much on the spatial information is also the main weakness of these methods. Natural images don't usually have a well defined object, in terms of the geometrical and positional symmetry of the objects. In fact, deep architectures have only been moderately successful in the natural image classification domain. Datasets such as Caltech 101 where deep representations have enjoyed any success have the objects centered in the image with the background cropped. This is demonstrated in figure 2.7. Most images have little or no clutter. The objects tend to be centered in each image. Most objects are presented in a stereotypical pose. Deep learning representations have struggled on more challenging image classification datasets such as Pascal VOC 2007. Figure 2.8 shows the Pascal VOC 2007 dataset. In this dataset, the objects vary a lot in scale and position; even multiple objects can be present in the same image.

Image representations such as BoW models, spatial pyramids and deep learnt representations all learn to recognize small artefacts in the image. BoW representations disregard all spatial context in the image and hence are very robust to scale and distortions, translations, rotations. However, they lose discriminative power in cases where spatial information is useful. Spatial pyramids are a means of incorporating weak geometry in BoW representations. They exploit spatial context by building BoW models for different regions in an image. Deep learnt representations are the other end of the extreme. They rely a lot on spatial information and thus lose some of the invariance to translations, rotations. Thus, the extent to which we rely on the spatial context is a crucial factor that affects the performance of any method.

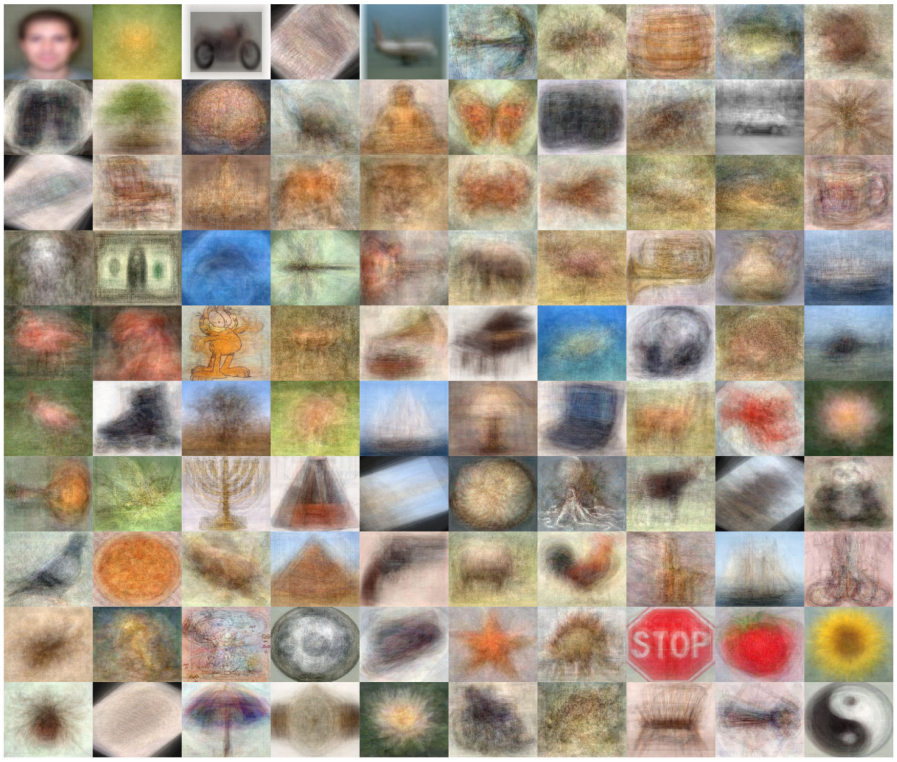


Figure 2.7: Caltech 101: composite image produced by averaging images of each category. Picture courtesy: Antonio Torralba.

Figure 2.8: Pascal VOC 2007 dataset. Sample images from a few categories.

Chapter 3

Partial Least Squares Kernel for computing similarities between Video Sequences

Computing similarities between data samples is a fundamental step in most computer vision (CV) tasks. Better similarity measures lead to more accurate prediction of labels. Computing similarities between video sequences has been a challenging problem for the CV community for long because videos have both spatial and temporal context which are hard to capture. In this chapter, we describe a novel approach that employs Partial Least Squares (PLS) regression to derive a measure of similarity between two tensors (videos). We demonstrate the use of this tensor similarity measure along with SVM classifiers to solve the tasks of hand gesture recognition and action classification. We also show that our methods significantly outperform the state of the art approaches on two popular datasets: Cambridge hand gesture dataset and UCF sports action dataset. Our method requires no parameter tuning.

3.1 Introduction and Prior Work

Many computer vision tasks involve assigning labels to unlabeled samples. Traditionally, we solve these tasks by acquiring ground truth labels for some of the data samples. Some measure of similarity between the seen and unseen samples is then used to predict the labels of the unseen samples. The similarity measure used is thus a crucial component of any CV problem. A lot of similarity kernels for real valued and symbolic data such as text can be found in literature [63]. However, there are few similarity kernels for videos (tensors). Devising good discriminative kernels for videos is challenging because they have both spatial and temporal context. In this chapter, we take a step forward in this direction.

Quantitative similarity measures between videos can be applied to solve various computer vision tasks such as hand gesture recognition and action classification. These find applications in Human Computer Interaction (HCI) [50] and video surveillance [61]. Hand gesture recognition is also widely used for sign language interpretation [22]. Studies have been conducted over the years to develop systems that perform these tasks accurately.

Some of the earlier methods for hand gesture recognition have used neural networks to recognize spatio-temporal actions [67]. Others describe videos using spatial [50] and temporal models [11]. Few methods have also used Hidden Markov Models and its variants [79]. More recently, graph matching approaches have been used for gesture recognition [62]. The most notable recent approach to hand gesture recognition [30] combines Canonical Correlation Analysis (CCA) with discriminant functions and SIFT features to extract discriminative pair-wise spatio temporal features (for pairs of videos) that perform robust gesture recognition.

There has been a furore of activity in the action classification community too. Methods that use the knowledge of the geometry of the tensor space [45] for action classication factor tensors using a modified High Order Singular Value Decomposition (HOSVD) and each factorized space is recognized as a Grassmann manifold; and classification is done on this manifold. Motivated by this approach, [44] represents tensors (videos) as a tangent bundle on a Grassmann manifold and canonical distances between these tangent spaces are then used for action classification. CCA has been extended [31] for multidimensional data arrays to inspect joint space-time linear relationships of two videos and acquire similarity features of the two videos that are both flexible and descriptive. This is achieved by representing third order tensors (videos) as a set of 2-D matrices and using CCA on each of these matrices. This method further uses a discriminative feature selection scheme and a nearest neighbour classifier for action classification.

Our method is similar to [31] in the sense that we too flatten the videos (third order tensors) to get three matrices (second order tensors) per video (these three matrices are referred to as the three joint shared modes of a tensor in [31]). However, unlike [31] that uses CCA, we use PLS regression to compute similarity between the corresponding second order tensors of a video. Finally, we build classification kernels using these similarity measures and use an SVM for classification.

3.2 Partial Least Squares

PLS [58] is a technique for modeling relations between sets of observed variables using latent variables. PLS assumes that observed data is generated by processes that use latent variables. PLS generates orthogonal score vectors (latent vectors) using the existing correlations between two sets of random variables while preserving most of the variance of both sets. PLS assumes that observed data is generated by processes that use latent variables and generates orthogonal score vectors (latent vectors) using the existing correlations between two sets of random variables while preserving most of the variance of both sets. The key difference between PLS and CCA (discussed above) is that CCA maximizes the correlation while PLS maximizes the covariance between two sets of variables.

Although PLS can tackle sets of random variables with different dimensionalities, our demonstration uses same sized random variables. Let $\mathbf{X}$ and $\mathbf{Y}$ be two sets of observed random variables (data). In our case, both $\mathbf{X}$ and $\mathbf{Y}$ are matrices of size $n \times m$ where n is the number of random variables and m is the dimensionality of each random variable. It is to be noted that both $\mathbf{X}$ and $\mathbf{Y}$ are preprocessed to

ensure they are both zero mean matrices. PLS models the relations between these two data matrices by decomposing them into:

$$\mathbf{X} = \mathbf{T}\mathbf{P}^{\mathbf{T}} + \mathbf{E} \tag{3.1}$$

$$\mathbf{Y} = \mathbf{U}\mathbf{Q}^{\mathbf{T}} + \mathbf{F} \tag{3.2}$$

where $\mathbf{T}$, $\mathbf{U}$ are $n \times p$ matrices containing p extracted latent vectors (also called scores), $\mathbf{P}$ and $\mathbf{Q}$ are $m \times p$ matrices of the loadings while $\mathbf{E}$ and $\mathbf{F}$ are the $n \times m$ matrices of residuals. In PLS regression, a linear inner relation between $\mathbf{U}$ and $\mathbf{T}$ is assumed:

$$\mathbf{U} = \mathbf{T}\mathbf{B} + \mathbf{H} \tag{3.3}$$

where $\mathbf{B}$ is the $p \times p$ diagonal matrix of regression coefficients. $\mathbf{H}$ is the matrix of residuals. Hence, equation (3.2) can be rewritten as:

$$\mathbf{Y} = \mathbf{T}\mathbf{B}\mathbf{Q}^{\mathbf{T}} + (\mathbf{H}\mathbf{Q}^{\mathbf{T}} + \mathbf{F}) \tag{3.4}$$

The sum of the regression coefficients in $\mathbf{B}$ serves as the quantitative measure of similarity between sets X and Y.

The PLS method, which is most commonly implemented using the nonlinear iterative partial least squares (NIPALS) algorithm [81], constructs a set of weight vectors $W = \{w_1, w_2, \ldots, w_p\}$ such that

$$[cov(t_i, u_i)]^2 = \max_{|w_i|=1} [cov(\mathbf{X}w_i, \mathbf{Y})]^2 \tag{3.5}$$

where t_i, u_i are the i^{th} column of matrices $\mathbf{T}$ and $\mathbf{U}$ respectively and $cov(t_i, ui)$ is the sample covariance between latent vectors t_i and u_i. After the extraction of t_i and u_i, the matrices X and Y are deflated by subtracting their rank-one approximations based on t_i and u_i. This process is repeated until convergence.

3.3 PLS Similarity Kernels for Videos

3.3.1 Joint Shared Modes

In section 3.2 we described a similarity measure between two sets of random variables. However, our goal is to classify videos, which are third order tensors. Thus, we need a way to convert the third order tensors to matrices (which are second order tensors). We achieve this by flattening the video tensor to a matrix. To understand this, we first consider the 2-D case: a matrix can be flattened to a 1-D vector by a simple row-wise (or column-wise) ordering of its elements. In the same way, a third order tensor can be flattened to a matrix in three ways, depending on which two dimensions are reordered into a 1-D vector.

A third order tensor $\mathbf{V} \in \mathbb{R}^{x \times y \times t}$ can be seen as a three dimensional matrix with three modes (dimensions): axes of space (x and y) and time (t). Assuming that we have videos of uniform size ($x \times y \times t$), as described above, there are three ways to flatten the video into a matrix: by re-ordering x, y or x, t or y, t. Thus, for any video, there are three distinct corresponding sets of matrices or random

variables. These corresponding matrices have been referred to as the joint shared modes of a tensor in [31]. We call these the xy, xt and yt joint shared modes and denote them by $\mathbf{V}_{xy}$, $\mathbf{V}_{xt}$ and $\mathbf{V}_{yt}$ respectively. Intuitively, by using three different joint shared modes, we are trying to encode the 3-D spatial and temporal context into 3 sets of random variables.

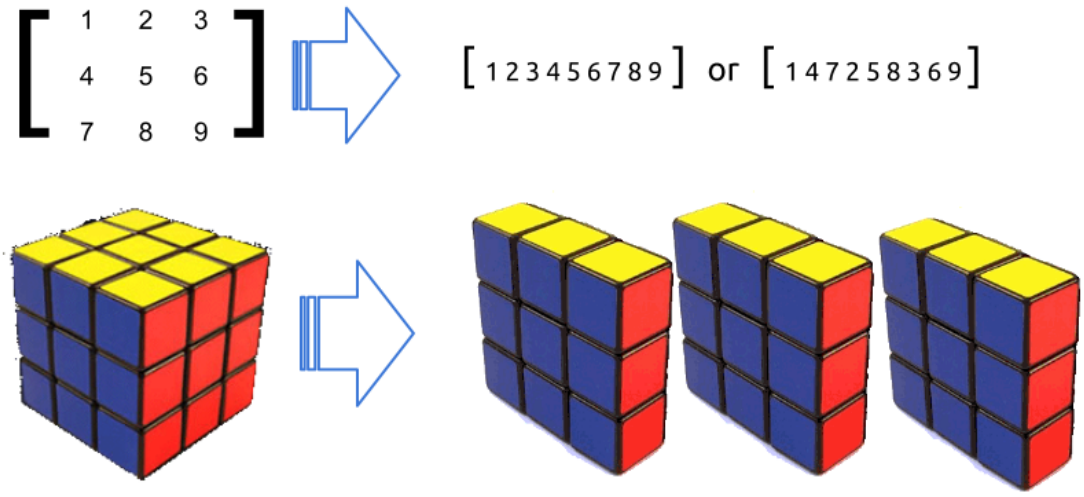


Figure 3.1: Flattening Videos to Matrices: A 2-D matrix can be flattened to a 1-D vector by a simple row-wise or column-wise reordering of the elements. We use a similar idea to flatten a 3-D video to 2-D matrices. For simplicity, a 3-D video can be seen as a rubic's cube. This cube can be cut into 2-D slices, and these slices can be stacked together in a 2-D plane, giving a two dimensional representation of the video. There are three axes along which we can slice the cube; Hence there are three ways of flattening a video. We call each of these 2-D representations of a video as a joint shared mode.

3.3.2 PLS Kernel

The PLS Kernel $\kappa(\mathbf{V}, \mathbf{W})$ gives a quantitative measure of similarity between two videos $\mathbf{V}$ and $\mathbf{W}$. As described in section 3.2, the quantitative similarity between two matrices (sets of random variables) is given by the sum of the regression coefficients in the diagonal matrix $\mathbf{B}$. We denote this similarity between two matrices $\mathbf{P}$ and $\mathbf{Q}$ by $\beta(\mathbf{P}, \mathbf{Q})$.

Each joint shared mode is treated as a set of random variables and contributes to the overall similarity between two videos. We compute the PLS regression coefficients between the corresponding modes for each pair of videos in the dataset. Since we have three joint shared modes corresponding to a video, essentially we can find three similarity values between each pair of videos, one corresponding to each joint shared mode. The PLS Kernel is given by:

$$\kappa(\mathbf{V}, \mathbf{W}) = \beta(\mathbf{V}_{xy}, \mathbf{W}_{xy}) + \beta(\mathbf{V}_{xt}, \mathbf{W}_{xt}) + \beta(\mathbf{V}_{yt}, \mathbf{W}_{yt}) \tag{3.6}$$

Thus, the similarity between two videos is simply the sum of the similarities between their corresponding joint shared modes.

3.3.3 Discussion

PLS regression is an extension of the multiple linear regression model on which a number of multivariate methods such as discriminant analysis, principal components regression, and CCA are based. Multivariate methods impose two restrictions: (a) latent variables are computed using the $\mathbf{X}^T\mathbf{X}$ and $\mathbf{Y}^T\mathbf{Y}$ matrices; cross-product matrices of $\mathbf{X}$ and $\mathbf{Y}$ variables are not used, and (b) the number of prediction functions is always smaller than the number of $\mathbf{X}$ and $\mathbf{Y}$ variables. In contrast, PLS extracts prediction functions from the $\mathbf{Y}^T\mathbf{X}\mathbf{X}^T\mathbf{Y}$ matrix. The number of prediction functions may be more than the number of $\mathbf{X}$ and $\mathbf{Y}$ variables. PLS can thus be used when the predictor variables outnumber the observations, unlike traditional multivariate methods.

3.4 Experiments and Results

We now discuss applications of PLS kernels to two popular CV tasks: hand gesture recognition and activity classification. Here we demonstrate the superiority of PLS similarity kernels over state-of-the art approaches on thest two tasks.

3.4.1 Hand gesture recognition on Cambridge dataset

The popular Cambridge hand gesture data set [31] contains 900 video sequences of 9 gesture classes, defined by 3 primitive hand shapes and 3 primitive motions (see Figure 3.2). Each class contains 100 video sequences; these 900 video sequences are partitioned into five different illumination setting subsets: $Set1$, $Set2$, $Set3$, $Set4$, $Set5$, each containing 180 videos. As in [44], we reduce the size of the video frames to 20×20 pixels and extract the middle 32 frames for classification. Thus, all the video sequences in the dataset were resized to $20 \times 20 \times 32$. The experimental protocol followed in [44, 31, 45] was used. According to this protocol, $Set5$ was used for training while $Set1$, $Set2$, $Set3$, $Set4$ were used for testing.

Training involved first computing the PLS Kernel Matrix containing the similarities between every pair of training video tensors. We used this Kernel Matrix to train a one-vs-rest SVM classifier [13] per gesture class. Testing involved computing the PLS Kernel Matrix containing the similarities between every pair of a training sample video and a testing sample video. This kernel matrix was used to generate SVM scores for each test sample. The test sample was assigned the class label of the classifier that gave the maximum score.

The hand gesture recognition accuracies can be seen in Table 3.1. We compare our method with the state-of-the-art approaches described in [44, 45, 31, 30] (section 3.1). Our method significantly outperforms the other methods on all illumination settings.

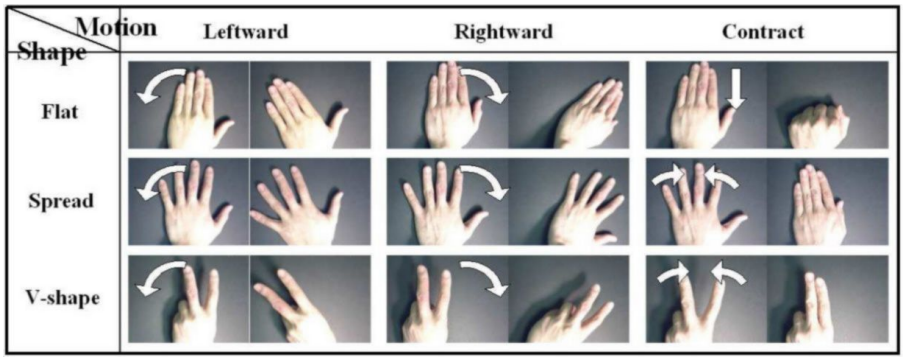


Figure 3.2: Cambridge hand gesture dataset

3.4.2 Action classification on the UCF Sport dataset

The UCF sport action dataset [57] contains 150 video sequences partitioned over ten human action categories like driving, kicking, walking, swinging golf clubs (see Figure 3.3). Each category has a different number of videos, from 6 to 22. This dataset is challenging because of the non-uniform backgrounds and relative motion between the camera and subject in some actions.

Figure 3.3: UCF Sports Action dataset

As in [44], we resize all the video sequences to the same size $32 \times 32 \times 64$. We choose the 64 middle frames from each video, and apply linear interpolation between frames for videos with less than 64 frames. We use the leave-one-out cross validation protocol just like in [44, 12, 33]. The classification setup remains the same as in our experiments on the Cambridge dataset. We trained a one-vs-rest SVM for each action class and the test video sequence was assigned the class label of the classifier with the maximum score. The classification results can be seen in Table 3.2. We have also compared results with [33] and [12]. While [33] learns the most discriminative space-time feature neighbourhoods for an

activity using local motion and appearance features, [12] computes rich features from point trajectories, combine local descriptors to combat background noise and use a novel feature selection scheme. Here again, our method significantly outperforms the state-of-the-art approaches.

Table 3.1: Hand-gesture recognition accuracy (%) on the Cambridge-Gesture Dataset

Method	Set1	Set2	Set3	Set4	Total
PLS	96%	92%	96%	93%	$\mathbf{94 \pm 2.1}\%$
TB [44]	93%	88%	90%	91%	$91 \pm 2.4\%$
PM [45]	89%	86%	89%	87%	$88 \pm 2.1\%$
DCCA [30]	-	-	-	-	$85 \pm 2.8\%$
TCCA [31]	81%	81%	78%	86%	$82 \pm 3.4\%$

Table 3.2: Leave one out cross validation on the UCF Sports Dataset

PLS	TB [44]	HDN [33]	OMD [12]
$\mathbf{93.2}\%$	88%	87.27%	86.9%

Discussion: Our PLS similarity kernel approach is superior to the previous best [44] for these tasks. Our method is both more intuitive (based on maximizing covariance) and straight-forward, thus easily implementable. Compared to [31], PLS is more general (sections 3.2, 3.3.3) and the use of SVM classifiers (as opposed to a nearest neighbour scheme) with our similarity kernels boosts the classification performance.

3.5 Summary

In this chapter, we devised a method that employs PLS regression to derive a scalar similarity measure between two sets of random variables. We extended this technique to find quantitative similarity measures between two videos. We employed discriminative kernel matrices constructed using pair-wise similarities between the data samples to solve the tasks of hand gesture recognition and human activity classification. Our method outperforms the state-of-the-art methods on the Cambridge hand gesture dataset and the UCF Sports dataset. Our model involves no parameter tuning.

Chapter 4

Learning Hierarchical Bag of Words using Naive Bayes Clustering

Image analysis tasks such as classification, clustering, detection, and retrieval are only as good as the feature representation of images they use. Much research in computer vision is focused on finding *better* or *semantically richer* image representations. Bag of visual Words (BoW) is a representation that has emerged as an effective one for a variety of computer vision tasks. BoW methods traditionally use low level features. In this chapter, we describe a novel strategy to use these low level features to create "higher level" features by making use of the spatial context in images. In this chapter, we propose a novel *hierarchical feature learning framework* that uses a *Naive Bayes Clustering* algorithm to convert a 2-D symbolic image at one level to a 2-D symbolic image at the next level with richer features. On two popular datasets, Pascal VOC 2007 and Caltech 101, we empirically show that classification accuracy obtained from the hierarchical features computed using our approach is significantly higher than the traditional SIFT based BoW representation of images even though our image representations are more compact.

4.1 Introduction and Prior Work

Over the years, research in computer vision has tried to narrow down the gap between raw image pixels and what humans see when they look at the image. The efforts to do so can be very broadly categorized into two classes. The first class of methods uses a robust representation based on relatively low level features (e.g. SIFT based Bag of Words (BoW) representations [65, 14]). BoW representations have received widespread success in a variety of computer vision tasks such as image classification and object detection [65, 14, 75] owing to their invariance to scale, spatial and rotational distortions. These methods also use domain knowledge. Over the years, much research has gone into improving the performance of models that employ BoW representations. Non-linear SVMs, specialized kernels of different kinds [52, 72, 78, 84], and spatial pyramids [37] have all contributed to the success of these representations. Most of these approaches tend to increase the overall image representation size. While these methods have been successful in capturing diversity in image patches at low levels, it is hard to

incorporate them into a hierarchical feature learning frameworks naturally as there is no systematic way to create higher level symbols from combinations of lower level discrete symbols (words).

To use the analogy from text data - representing images as histograms of SIFT [43] or HoG [15] based bag-of-words is like representing text documents as histograms of letters in the language. Semantically shallow features (e.g. lines in images or *letters* in text) are highly "ambiguous". A vertical line in isolation in one image can mean something completely different (e.g. a needle on a wall clock) from what it means in another image (e.g. spoke in a bicycle wheel). Comparing simple features such as the number of vertical lines in images or in a corresponding image quadrants reaches cannot address the fundamental problem of symbol ambiguity. The key to better computer vision systems is therefore in building a hierarchy of semantically deeper features (e.g. face, wall clock, chair, etc.) preferably in an unsupervised manner before doing various image analysis tasks. A hierarchical feature that uses neighboring low level features to create a higher level feature reduces ambiguity by incorporating "context". The "meaning" of a higher level feature is more than the sum of its parts. It is this enrichment that makes hierarchical feature learning paradigms more promising.

The second class of methods continue to *enrich* low level features by using *hierarchical feature learning paradigms*. Most hierarchical feature learning approaches such as Deep Belief Networks [26], Convolutional Neural Networks [39], Convolutional Deep Belief Networks [40] and other hierarchical models [56] use simple building blocks such as a logistic function to learn complex overall models with many parameters to tune. Ideas like layered incremental training have made the training of these models practical. Further they can model translation invariance well using max pooling. The limitation in these models is that they need real-valued inputs at each layer in the hierarchy and hence the traditional BoW approaches that generate a large number of discrete symbols at lower levels cannot be naturally incorporated into such hierarchical feature learning frameworks. Also, these models are complex with many parameters to tune. While the initial results on these look promising, it is prohibitively expensive to train these models as there are many parameters to learn and many ways to fall into a local minima (initialization, update schedule, learning rates and momentum terms, etc). Since *"seeing" is the process of translating real valued 2-D pixel maps into "discrete" objects*, methods that depend on real-valued inputs are not naturally suitable for building a hierarchy of discrete objects. While the deep learning methods have been known to work well for a variety of simpler datasets such as ILSVRC 2010, ImageNet and Hollywood 2, they haven't enjoyed much success on more challenging datasets such as PASCAL [20].

Research into bridging the gap between these two directions exists. Hyperfeatures [2] exploit the spatial co-occurrence statistics at scales larger than their local input patches by aggregating local descriptors using methods such as GMM and LDA. This work is intended as a step forward in this direction. We propose to combine the strengths of both the hierarchical feature learning and the Bag-of-Words learning paradigms. We propose a generic framework for building *discrete feature hierarchies* in an unsupervised fashion starting from any first level symbol image (e.g. dense SIFT visual words). The framework has two parts: First is a novel *Naive Bayes Clustering* algorithm that clusters symbolic image patches using

EM like updates to maximize the log likelihood of the data in terms of a mixture of naive Bayes discrete multi-variate distributions. Second we do a maximum pooling on neighboring patches using the posterior probabilities of clusters in data points to reduce the image size at the next level. Evaluations on Caltech 101 and Pascal VOC 2007 show significant improvements in head to head comparisons between traditional SIFT based BoW (level 1) and image representations from hierarchical features learnt by our framework. We argue that our representation is both compact and semantically meaningful.

4.2 Background

In this section we briefly summarize the two prominent directions in computer vision both of which endeavour to represent the visual world through features which are invariant to scale, translation, rotation, illumination, occlusion, and so on. However, while the BoW approaches can improve by further exploiting the information content in the spatial layout of images, the deep learning methods can enhance their utility by overcoming the training and architectural complexity for learning large scale computer vision systems. We intend to learn from these two directions and take an approach that combines their powers to achieve superior representation.

4.2.1 Beyond Bag of Words

Visual BoW draws its inspiration from the analogous BoW models for document representation that ignore the order of words. Traditional image classification involves computing local features at interest points in an image and pooling these local features to give a global image representation. BoW essentially quantizes each local feature into one of the visual words using a codebook and then represents each image as a histogram of visual words. Computing the codebook involves identifying interesting local patches in an image, extracting features or keypoints such as SIFT from these local patches and finally clustering (usually using K-means) to group key points from the training images into clusters; the center of each cluster corresponds to a different visual word. Finally each SIFT vector is quantized by assigning it the label of the nearest cluster center. We represent each image as a histogram of the visual words, called the BOW representation.

BoW represents an image using the distribution of visual word occurrences. In doing so, it converts images of different sizes into fixed length representations. This is especially convenient for the classification task that need fixed dimensional inputs. However, BoW relies only on the appearance of the visual words and ignores their spatial layout. This characteristic imparts invariance to scale, translation and deformation, at the cost of discriminative power especially when the spatial layout is important.

There have been many recent attempts to overcome the limitations of BoW [53]. These include part generative models like [19] and frameworks that use geometric correspondence search [36]. These work well but are computationally expensive. BoW can be enhanced [64] by extending the codebook to include *doublets* which are pair-wise relations between features that lie in the same local neighbourhood.

Spatial pyramids [37] was a major breakthrough in this direction; it incorporates spatial information by computing bag of word representations for different image regions at different scales and concatenating these representations and finally uses a pyramid matching kernel [23] for classification. Almost everything in the book - from kernels [52, 72, 78, 84] to sparsity [82] to local codes [78] has been attempted to enhance the power of these low level representations [29].

All these clearly indicate the recognition of raising the semantic depth of the low level features discovered through the BoW process. Bringing context of neighboring features to define "higher level" features is clearly recognized as the next natural step here. As described in section 4.1, hyperfeatures [2] were devised especially to fulfil this need. In this chapter, we continue to explore this middle ground.

4.2.2 Deep Learning

Deep learning networks [26, 40] and convolutional networks [39] represent an orthogonal school of thought. These are driven by the idea that good internal representations are hierarchical and can be learned directly from the data. These networks have multiple hierarchical layers (also called *feature maps*) stacked together; each feature map learns artifacts in the image by assembling smaller artifacts learnt by the preceding feature maps. Pixels are assembled into edges, edges into object parts, object parts into objects, and objects into a scene; deep learning thus exploits the spatial information in the images. These levels represent the feature hierarchy.

Convolutional networks typically work on *overlapping* patches at each level (the overlap takes care of small translations) and summarize the features learnt in a neighbourhood by a pooling method. Popular pooling methods are (a) average pooling where the features computed over a region (called a cell) are averaged to give the feature representation of the cell and (b) max-pooling where the feature representation of the cell is the maximum of the features in the cell. Convolutional networks usually alternate between feature maps and pooling layers to achieve invariance to small translations and distortions.

Deep learning gives robust image representations. However *insufficient depth can hurt*. Also, training deep networks involves making many design decisions, huge training set, it is computationally challenging and most of the feasible training algorithms are mostly approximations of the actual objective. (In fact, attempts at training deep networks had failed before [26].)

There have been a few attempts to bridge the gap between the two schools by taking the middle ground and developing frameworks that exploit the advantages of both [34]. This work is another endeavour towards this objecive. Our approach involves deep / hierarchical learning of higher level discrete symbols from lower level discrete symbols (for instance BoW visual words) that lie in the same spatial neighbourhood. Our approach is different from traditional deep learning in that we work with symbols / visual words and not real valued features. A novel Naive Bayes Clustering method allows us to cluster combinations of low level symbols.

4.3 Naive Bayes Clustering

In order to build hierarchy of discrete features to compose symbols at the next level using the right juxtapositions of symbols at the previous level, we need a systematic way of dealing with the combinatorial explosion. For example, if we use 1000 low level features obtained from BoW and create higher level symbols from just 2×2 patches of SIFT visual words, the potential combinatorial space of discrete symbols at next level is O(10^{12}), clearly too prohibitive to just do traditional 2×2-gram histogram counting. If this were real-valued data, we could use any clustering technique but since this is symbolic data in a large vocabulary we need to use a non traditional clustering technique.

In this section we present a novel Naive Bayes clustering algorithm to cluster multi-variate discrete data in general and discrete image patches in particular. Note that in our experiments, we start with SIFT-BoW visual words; a discrete image patch is thus a patch of such visual words. To define a clustering algorithm, we need to define a "cost function", a "cluster representation" and of "update rules" to learn the cluster centers and cluster associations with data. First some notation: Let $\mathbf{X} = \{\mathbf{x}^n = (x_1^n \dots x_D^n)\}_{n=1}^N$ be the set of N data points. Each feature $X_d \in \mathbf{V}_d$ comes from a *discrete* feature vocabulary $\mathbf{V}_d = \left\{v_1^d \dots v_{M_d}^d\right\}$ of size $M_d = |\mathbf{V}_d|$. In image domain, each 2-D discrete image patch of size $P \times P$ is treated as a one-dimensional vector of size $D = P^2$ and each symbol comes from the same vocabulary, (i.e. $\mathbf{V}_d = \mathbf{V}$ of dense SIFT clusters.)

4.3.1 Mixture of Multi-variate discrete Naive Bayes

Mixture models [8] are commonly used to partition the data into meaningful clusters. Our patches are in $P \times P$ discrete space. Typically a parametric mixture model is learnt by maximizing a maximum (log) likelihood objective over the data:

$$J(\mathbf{\Theta}) = \log \prod_{n=1}^{N} P(\mathbf{x}^n) = \sum_{n=1}^{N} \log \sum_{k=1}^{K} P(k) P\left(\mathbf{x}^n | k\right). \tag{4.1}$$

Depending on the nature of the data, the mixture density function $P(\mathbf{x}|k)$ takes different forms. For example when $\mathbf{x} \in R^D$ any real-valued multi-variate density function such as a full Gaussian can be used. In our case $\mathbf{x} \in \mathbf{V}^D$ and therefore, we propose to use the simplest multi-variate discrete density function, i.e. *Naive Bayes* (NB):

$$P(\mathbf{x}^n|k) = \prod_{d=1}^{D} P\left(x_d^n|k\right) \tag{4.2}$$

In NB clustering, therefore we learn a "mixture-of-Naive Bayes" parametric generative model (Eq. 1) over a multi-variate discrete data by conveniently assuming independence among the features (Eq. 2). In general there are two constraints that are also part of the objective. The priors must add up to one and the density functions over all possible values that each feature can take for any given mixture component

must also add up to one, i.e.,

$$\sum_{k=1}^{K} P(k) = 1, \sum_{m=1}^{M_d} P\left(v_m^d|k\right) = 1, \forall d = 1 \ldots D \tag{4.3}$$

A total of $K \times \left(1 + \sum_{d=1}^{D} M_d\right)$ parameters $\mathbf{\Theta} = \left\{P(k), \left\{P(v_m^d)\right\}_{m=1}^{M_d}\right\}_{k=1}^{K}$ are learnt using an EM-algorithm with the following update rules for the E-step (Eq. 4.4) and smoothed M-step (Eq. 4.5 and 4.6) from iteration $t-1$ to iteration t.

$$P_t(k|\mathbf{x}_n) = \frac{P_{t-1}(\mathbf{x}_n|k)P_{t-1}(k)}{\sum_{k'}^{K} P_{t-1}(\mathbf{x}_n|k')P_{t-1}(k')} \tag{4.4}$$

$$P_t(k) = \frac{\lambda + \sum_{n=1}^{N} P_t(k|\mathbf{x}_n)}{\lambda K + N} \tag{4.5}$$

$$P_t(v_m^d|k) = \frac{\lambda' + \sum_{n=1}^{N} \delta(x_{n,d} = v_m^d) P_t(k|\mathbf{x}_n)}{\lambda' M_d + N P_t(k)} \tag{4.6}$$

Equation 4.4 computes the posterior probability of assigning a data point to cluster k in the next iteration (t) given the parameters at the previous iteration ($t-1$). Equations 4.5 and 4.6 are the parameter updates based on the assignment of data points to the clusters in this iteration. δ is the *kronecker delta*. Here we employ basic laplacian smoothing that takes affect mostly if the number of points in a cluster is small compared to the vocabulary size.

4.3.2 Soft vs. Hard Clustering

The EM algorithm described above represents a soft clustering algorithm where each data point is assigned to all clusters using the posterior probabilities i.e., $P_t(k|\mathbf{x}_n)$ in each iteration. This increases the computational complexity of the other update rules by a factor of K. In traditional (hard) clustering in each iteration, a data point is assigned to the cluster with the highest posterior probability. The hard clustering version of the above soft clustering algorithm alternates between the assign cluster E-step (Eq. 4.7) and the cluster parameters M-step (Eq. 4.8 and 4.9):

$$\kappa_{t-1}(\mathbf{x}^n) = \arg\max_{k=1\ldots K} \left\{P_{t-1}(\mathbf{x}_n|k)P_{t-1}(k)\right\} \tag{4.7}$$

$$P_t(k) = \frac{1}{N} \sum_{n=1}^{N} \delta\left(\kappa_{t-1}(\mathbf{x}^n) = k\right) \tag{4.8}$$

$$P_t\left(v_m^d|k\right) = \frac{\sum_{n=1}^{N} \delta\left(x_d^n = v_m^d\right) \delta\left(\kappa_{t-1}(\mathbf{x}^n) = k\right)}{\sum_{n=1}^{N} \delta\left(\kappa_{t-1}(\mathbf{x}^n) = k\right)} \tag{4.9}$$

Hard clustering is faster since parameter updates take K times less time per iteration. Combined with a smarter initialization strategy discussed below, we found this to be better than soft clustering in terms of convergence and quality.

4.3.3 Smart Initialization

Sensitivity to initialization is a well known problem with clustering. Bad random initializations typically result in slow convergence, poor clustering quality and require multiple runs with different random initializations to generate the right final clusters. This randomness and uncertainty in clustering initialization can be mitigated by a number of smart initialization strategies [3]. In this chapter, we employ a *farthest first point* (FFP) initialization. The goal of this initialization is to pick the initial K clusters such that they "cover" the entire data space well by spreading themselves as far away from each other as possible. Representation score of a point is defined as the similarity of a data point with the nearest cluster. The similarity between two data points $\mathbf{x}^n$ and $\mathbf{x}^{n'}$, $sim(\mathbf{x}^n, \mathbf{x}^{n'}) = \sum_k \delta(x_d^n = x_d^{n'})$ The FFP algorithm works as follows:

1. Initialize:
   - First cluster randomly: $k \leftarrow 1, \mu_1 = \mathbf{x}^r$ where r =random$(\{1 \dots N\})$
   - *Representation scores*: $R(\mathbf{x}^n|\mu_1) = sim(\mathbf{x}^n, \mu_1),\ \forall n = 1 \dots N$
2. Sample least represented point as the next cluster. If there are more than one equally representative points, pick one randomly.:

$$\mu_{k+1} = \arg \min_{n=1\dots N} R(\mathbf{x}^n|\mu_1 \dots \mu_k) \tag{4.10}$$

3. Update the representation scores of all data points:

$$R(\mathbf{x}^n|\mu_1 \dots \mu_{k+1}) = \max\{R(\mathbf{x}^n|\mu_1 \dots \mu_k), sim(\mathbf{x}^n, \mu_{k+1})\}, \forall n = 1 \dots N$$

4. $k \leftarrow k + 1$, repeate steps 2 through 4 while $k < K$

FFP based smart initialization gives significantly better clusters and faster convergence than traditional random initializations.

4.4 Learning hierarchical bag of words

Any form of vector quantization gives a symbolic representation to the keypoints. Kmeans as a vector quantization framework has the limitation that it can cluster real valued keypoints only, because it has no distance metric to compare symbols. This is the primary hurdle that has prevented the evolution of models that learn hierarchical bags of features. As described in section 4.3.2, the NB clustering algorithm is designed to cluster symbolic data and hence it can be used to quantize discrete symbolic vectors. With this useful tool, we are prepared to exploit the principles of deep learning to learn features in the BoW domain.

4.4.1 Approach

We start conventionally by employing K-means on features computed at local image patches to give us symbol representations for the low level keypoints. Given these representations, we compute BoW representations of the images: we refer to these as our first level image representations / features. However, we do not lose the symbolic image yet, for it has spatial context. Adhering to the conventional mode of feature extraction, we collect keypoints (vectors of symbols) from patches in a dense grid over the level 1 symbol image. We quantize these symbolic vectors using the naive bayes clustering approach to get another level of symbols and another symbol image in turn. The symbols at this level are aggregations of the symbols at the previous level that lie in the same local neighbourhood. We compute the BoW representation of these level 2 symbol images and call these the second level image representations. We have thus devised a hierarchical feature extraction scheme that is independent of the we get the visual words at any level: this process can be repeated any number of times to get a desired level of image representation. Figure 4.1 describes our approach.

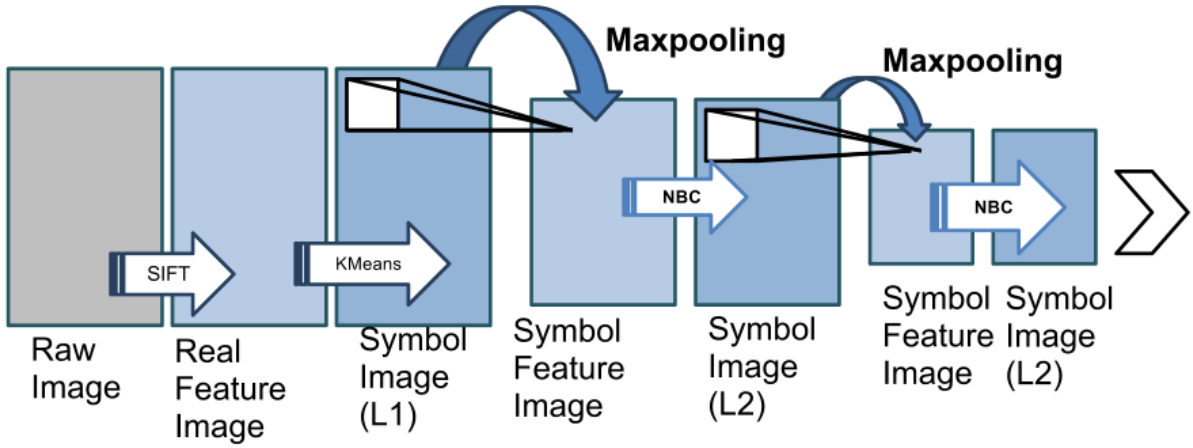


Figure 4.1: Block diagram of our approach. SIFT features are computed on the raw image patches and quantized using K-means to get the first level symbol image. Henceforth, keypoints at any level of the hierarchy are collected from patches in a dense grid over the symbol image at the previous level. These keypoints are clustered using NB clustering and quantized to get the the symbol image at the current level. This process can be repeated any number of times. BoW representations can be computed using the symbol image at any level of the hierarchy and used for classification.

4.4.2 Maximum Pooling

Spatial pooling is an idea borrowed from the deep learning community that introduces compactness in the representation and imparts invariance to distortions by reducing the spatial resolution. As described in chapter 2 (section 2.3), deep Learning methods typically compute features over overlapping patches in an image. This overlap takes care of small translations of artefacts in the image. Overlapping causes the same information to be captured more than once, and increases the representation size.

Maximum pooling is a form of spatial sub sampling (section 2.3.1). Once a feature has been detected, its exact location becomes less important (and harmful). Only its approximate position relative to other features is relevant. A simple way to reduce the precision with which the positions of distinctive features are encoded is to reduce the spatial resolution, thus summarizing the features learnt in a neighbourhood by a pooling method. Pooling thus also reduces the representation size.

Conventionally, spatial pooling is done over a grid of cells where the keypoints within each cell are summarized by a single keypoint. For a cell c spanning $P \times P$ symbolic keypoints ($\mathbf{x}^n, n = 1, \ldots, P^2$), we define the cell representative α_c to be the symbol with the maximum posterior probability as given by

$$\alpha_c = \arg\max_n \{P(\mathbf{x}_n|\kappa(x^n))P(\kappa(x^n))\} \tag{4.11}$$

where $\kappa(x^n)$ is the cluster representative of x^n. In our experiments, we follow the usual convolutional network maxpooling protocol which uses non-overlapping patches of size 2×2 pixels.

4.5 Experiments, Results and Discussions

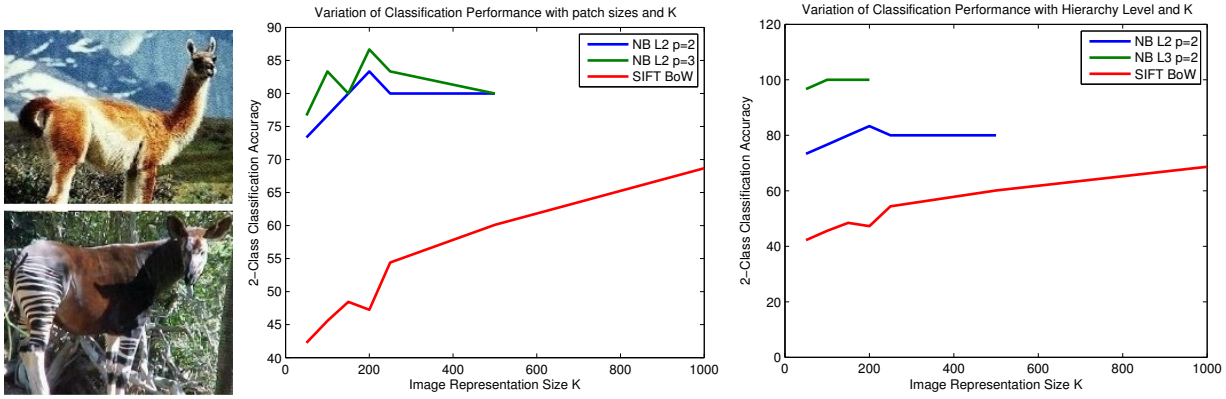
In this section, we use the NB clustering to learn hierarchical feature representations and use the learnt representations for the task of image classification. We first demonstrate our approach on a simple two class classification problem, and later show comprehensive results on two popular object classification datasets, namely Caltech 101 and Pascal VOC 2007. In this process, we gain insights into the learning by studying the effect of the parameters like the patch size (p), the size of the symbol space at each level (K) and the level of the hierarchy (l). We argue that the method learns semantically meaningful concepts by assessing the objective we are trying to achieve. We also demonstrate that hierarchical representations learnt through the NB clustering of visual words are better representations and outperform the traditional BoW method of image classification.

Experimental setup: In all the following experiments, we extract SIFT features over a dense grid using a scale of 12 and a shift of 6 pixels. Our baseline BoW representations are computed by clustering the SIFT vectors into 1000 visual words. For classification, we use a χ^2 homogeneous kernel map [75] on the BoW histograms and use a linear pegasos SVM [74].

4.5.1 Two Class Classification: Okapi vs Llama

For the first set of experiments, the two classes we work with are llama and okapi (Figure 4.2a) which are part of the Caltech 101 dataset. We sample 15 images randomly for training and testing each from both these classes. This gives us training and testing sets of sizes 30 images each. These classes are hard to differentiate because the two animals look structurally similar and are found against similar looking backgrounds.

For these experiments, we use our baseline BoW representations to compute Level 2 features using patch sizes $p_2 = 2, 3$ with shifts 1 and 2 respectively and vocabulary size $K_2 = 50, 100, 150, 200, 250, 500$



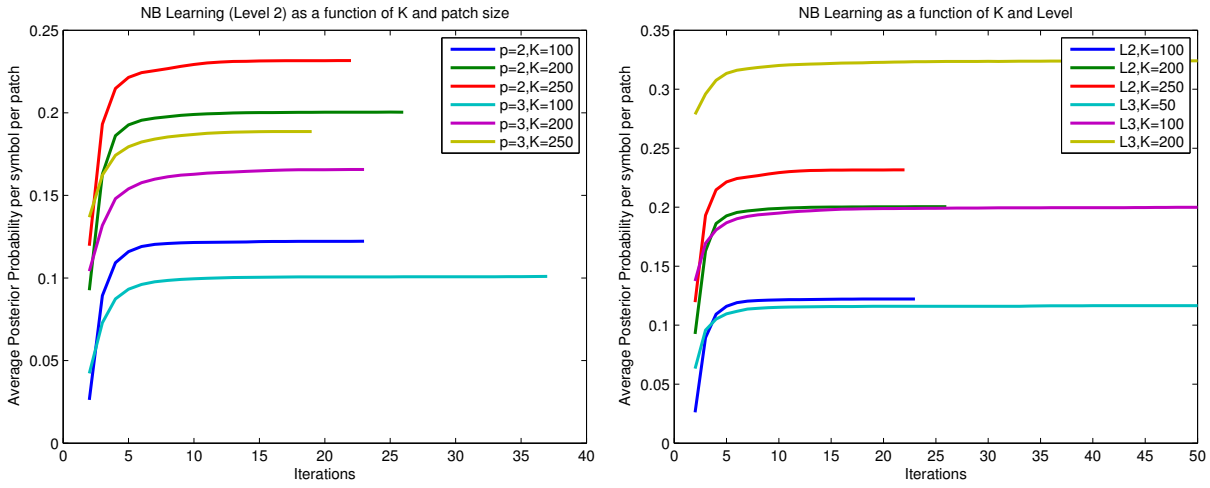
(a) (b) (c)

Figure 4.2: Two-Class (Llama vs Okapi) Classification. (a) Llama (top) and Okapi (bottom) (b)Variation of accuracy with level 2 patch size and size of symbol space. (c)Classification accuracy based on Level 2,3 features; Patch size was fixed to p=2 for these experiments.

(from this point on, we denote the patch size and the size of the symbol space at the n^{th} level by p_n and K_n respectively). BoW on the level 2 features given by each combination of p_2 and K_2 gives us a different level 2 representation. We compute Level 3 features using the level 2 representation given by $p_2 = 2, K_2 = 200$. For level 3, we use $p_3 = 2$ with shift of 1 and $K_3 = 50, 100, 200$. We use classification accuracy as the performance metric in our evaluations in these experiments.

Figures 4.2 (b) and (c) compare the classification performance of hierarchical representations with the baseline BoW. In (a) we fix the representation level ($l = 2$) and vary K_2 and p_2; in (b) we fix p_2 and vary the representation level ($l = 2, 3$) and K_2 and K_3. In (a), level 2 features significantly outperform level 1 features. Also, a patch size of 3 works gives better accuracy. At the representation size of 200, the performance gap between level 1 and level 2 features is 30%. In (b) The plots demonstrate the improvement in classification accuracies as we build higher representations. For level 3, we hit the 100% accuracy bound at $K_3 = 100$ while for level 1, accuracy is merely 68% for $K_1 = 1000$. Hence we achieve 32% higher accuracies using $1/10^{th}$ representation size.

Figures 4.3 (a) and (b) investigate what goes on at the core of the NB clustering algorithm during the vocabulary building procedure. We plot the average posterior probability per symbol per symbolic patch over the epochs of the training procedure. This is the average probability of a symbol to assume a particular position in a patch. This in turn determines the probability of a patch being part of a cluster of patches. In (c), we fix the level of representation ($l = 2$) while varying p_2 and K_2; in (d) we fix patch size and vary the representation level ($l = 2, 3$) and the number of clusters K_2 and K_3. It can be observed that bigger patches have lower average probabilities per symbol. This can be attributed to the fact that clustering a vector of 9 symbols is tougher than clustering a vector of 4 symbols because bigger patches are more complex in the number of ways the symbols are aligned in a patch. Another observation is that this probability increases as we increase the number of clusters. This can be explained by stating that increasing the number of clusters is allowing the arrangements of symbols in a patch more states to be



(a) (b)

Figure 4.3: (a) Plot of mean posterior probabilities per symbol per patch over epochs. Effect of the patch size (p) and size of symbol space (K) can be seen here. (b) NB Learning for different sizes of symbol space (K) across hierarchical levels 2 and 3. Higher probabilities for Level 3 show that the method is learning semantically meaningful concepts.

in. Thus, each patch is more likely to find a state that it is most similar to. Finally, we comment on the increase in these probabilities across levels of the hierarchy. Figure 4.3 (b) shows that the probabilities are higher for level 3 (for a fixed K). This shows that patches at this level are more likely (than patches at level 2) to find states / configuration of symbols that describe them. This has a direct bearing on the purity of the representations in terms of what they mean semantically. Referring back to the comment we made about the needle and the spoke in the bicycle wheel, the confusion between the two is resolved once we learn the difference between a clock and a bicycle.

4.5.2 Caltech 101

Caltech 101 contains a total of 9146 images, split among 101 distinct object categories. In these experiments, we sampled 30 random images for training from each of the 101 categories, getting a total of 3030 training images; the rest of the images were treated as testing images; however, as in [37], we limited the number of testing images per category to 50. These experiments were repeated 5 times with random subsampling and the mean classification accuracies over the five experiments are reported. To compute the BoW codebook, we sampled 5 training images from each category (505 images in all). We trained a one-vs-rest SVM for each class and the test image was assigned the label of the classifier with the highest score and report the accuracy of the classification.

For level 2, we use patches of sizes 2×2 and 3×3 with shifts of 1 and 2 pixels respectively. To compute the level 2 vocabulary, we sample 5 images randomly from each class and further sample each of these images to collect 25% of the total keypoints per image; we use $K_2 = 100, 250$. For the third

Table 1: Classification accuracies on Caltech 101 for combinations of Spatial Pyramid and NB hierarchical features (and the baseline BoW). Table 2: Classification Accuracies on Caltech. BoW represents our baseline BoW results, BoW* represents the results quoted in [37]. Note that [37] uses pyramid kernel (and we use χ^2) and different scale, shift for SIFT computation. NBC represents our best results corresponding to L2, $K_2 = 250$ with level 2 SPM.

Table 4.1: Caltech 101- NB + SP

SPM	BoW	L2(250)	L3(200)
L 0	$43.4 \pm 1.2\%$	$60.4 \pm 0.7\%$	$61.3 \pm 1.4\%$
L 1	$59.0 \pm 0.8\%$	$68.2 \pm 0.8\%$	$66.6 \pm 1.6\%$
L 2	$68.3 \pm 1.3\%$	$72.4 \pm 0.6\%$	$69.8 \pm 0.9\%$
L 3	$67.6 \pm 0.7\%$	$67.8 \pm 1.1\%$	$66.3 \pm 1.4\%$

Table 4.2: Classification on Caltech

Method	Accuracy
BoW* [37]	$64.6 \pm 0.8\%$
CDBN [40]	$65.4 \pm 0.5\%$
BoW	$68.3 \pm 1.3\%$
NBC	$72.4 \pm 1.8\%$

level, the patch size is 2×2 with a shift of 1 and $K_3 = 100, 200$, vocabulary is computed using 5 random training images per class and using 25% of the keypoints per image. The classification pipeline remains the same as in the baseline case.

The classification accuracies for these procedures can be seen in Tables 4.1 and 4.2. It can be seen that hierarchical features learnt using the NB clustering approach significantly outperform the baseline BoW representation. Table 4.1 compares the classification performance of the baseline BoW representation with features at levels 2 and 3 and also shows further improvement in classification performance by using spatial pyramids on top of the image representations derived through the various methods. Hence, the representative power of hierarchical features can be further enhanced by using spatial pyramids. Note that in these experiments, we use only the spatial pyramid representation and not the pyramid matching kernel. As mentioned earlier, we use a homogeneous χ^2 kernel map; In Table 4.2, there are two rows devoted to BoW. BoW represents our baseline experiments (with χ^2 kernel), BoW* reports the accuraries quoted in [37] (Note that [37] uses SIFT features computed at a scale of 16 and shift of 8, and a pyramid matching kernel).

It should be noted that while the dimension of the spatial pyramid representation increases many folds as we move a level up, our hierarchical representations typically become more compact. We achieve better classification performance despite this fact.

4.5.3 Pascal VOC 2007

Pascal VOC 2007 data set has a total of 9955 images, split into 5011 training and 4944 testing images, distributed across 20 object categories. We use the entire dataset for our experiments. In these experiments, the BoW codebook was computed by clustering the keypoints collected from 10 random training images from each category (200 images in all). The classification scheme here is different from our experiments on Caltech. Pascal dataset allows multiple object categories in the same image, hence computing classification accuracies by assigning the class label of the classifier that returns the highest score to the test image is not fair assessment. In this set of experiments, we train a classifier for each

class and compute Average Precision (AP) over the ranked list of test images. We finally report the mean AP over all the classes. Note that for our final image representation, we use a 2^{nd} level spatial pyramid [37]. As mentioned in section 4.1, deep learning methods have not enjoyed much success on this challenging dataset. Unlike Caltech 101, where the images are aligned and centered, Pascal has significant variation in the scale, position and orientation of the object in the image; it also allows multiple objects in the image. Note that CDBN [40] results are not available on Pascal 2007.

For level 2 features, we experiment with patch sizes of 2×2, 3×3 with shifts of 1 and 2 respectively. To compute the level 2 vocabulary, we sample 10 images randomly from each class and further sample each of these images to collect 25% of the total keypoints per image; we use $K_2 = 100, 250$. For level 3, we use patches of size 2×2 with a shift of 1 and $K_2 = 100, 200$. We use the same classification pipeline to classify the features at level 2 and 3. Table 4.3 displays the classification results for the mentioned methods. Here again, we significantly outperform the baseline results. While the baseline representation achieves a mean AP of 52.8% using a representation size of 1000, our L3 representation achieves 57% at a smaller representation size of 200. Hence, our representation is both richer and more compact.

Table 4.3: Classification Results on the Pascal VOC 2007 dataset. The table shows mean classification APs over 20 classes.

Method	SIFT BoW	L2	L2	L2	L3	L3	L3
p	-	3	3	2	2	2	2
K	1000	100	250	100	250	100	200
AP	52.84	54.90	55.86	55.64	56.20	56.48	57.04

4.5.4 Discussion

Our results on two classes explore the semantic meaning of the learnt hierarchical representations. Empirical comparisions with spatial pyramids reveal that we can achieve better classification accuracies by using a simpler kernel and with a much smaller feature / representation size. Both these methods endow BoW with means to use spatial context. However, there is a fundamental difference in both approaches: while spatial pyramids intend to discover the same low level artifacts in different regions of the image, we learn aggregates of such low level artifacts in the hope that these artifacts are part of a larger context and repeating this process of aggregation over and over will eventually lead us to learning the objects we are trying to classify.

By empirically outperforming both SPM and CDBN representations on the two datasets, we have tried to demonstrate that our representations are not only richer but also more *compact* in size. For example, in Table 4.3, our L3 representation of size 200 outperforms the baseline representation of size 1000 significantly. The performance of our approach can be further improved by allowing a larger

representation size, i.e. by using multiple scales of SIFT features as in [29]. In these experiments, our primary focus was to demonstrate the superiority of our representation over traditional BoW.

Finally a note on the the speed of the algorithm. Naive Bayes Clustering is typically slightly less than an order slower than K-Means, simply because of the number of operations in each iteration. Any algorithmic solution is characterized by several parameters: the accuracy of the solution, the simplicity, the resources it consumes, and the processing time. In this chapter, we have endeavoured to improve the accuracy of our solutions by building better representations, because typically most of our learning is done offline, and not in a real time setting. Having said that, we acknowledge that the performance gains we achieve come at computational costs. Our methods is iterative, and there are several algorithmic parameters (the patch sizes, the learning rates etc) to tweak and tune. This tuning requires parameter sweeps.

4.6 Summary

In this chapter we devised a clustering framework for symbolic data points which can be used to learn hierarchical features starting with discrete data (such as BoW symbols.) Our method attempts to bridge the gap between two directions of research by developing a framework that learns from both approaches. We produce experimental evidence to argue that our hierarchical representations are semantically meaningful. We back this claim by outperforming the traditional BoW and deep learning representations on popular image classification datasets. It is quite possible that there are better distance functions between discrete features that may improve the learning procedure and the representative power of such a framework.

Chapter 5

Learning Multiple Non-linear Subspaces using K Restricted Boltzmann Machines

The overall complexity in building descriptive or discriminative models is shared between the features derived from raw data and the models that use these features as inputs. Simple features require complex models while more sophisticated features require simpler models to achieve the same level of model quality. Learning semantically richer features is, therefore, the key to building simpler, more interpretable, and more accurate models. In domains such as images, where the data (image patches) might lie in multiple non-linear manifolds, feature learning becomes even more important. In this chapter, we propose a framework that uses K Restricted Boltzmann Machines (K-RBMs) to learn non-linear manifolds in the raw image space. We solve the coupled problem of finding the right non-linear manifolds in the input space and associating image patches with those manifolds in an iterative Expection Maximization (EM) like algorithm to minimize the overall reconstruction error. Extensive empirical results over several popular image classification datasets show that such a framework outperforms the traditional feature representations such as the SIFT based Bag-of-Words (BoW) and convolutional deep belief networks.

5.1 Introduction and Prior Work

Feature extraction and modelling together address the overall complexity of mapping the raw input to the final output in modelling. Rich features that capture most of the complexity in the input space require simpler models while simpler features would require more complex models. This "law-of-conservation of complexity" in modelling has driven many efforts in feature engineering, especially, in complex domains such as computer vision where the raw input is not easily tamed by simple features. Finding semantically rich features that capture the inherent complexity of the input data is one of a challenging and necessary pre-processing step in many machine learning applications.

We propose a feature learning framework that uses the hypotheses: data really lies in multiple non-linear manifolds and finding those manifolds and clustering the right data points into the right manifolds

will result in the kind of features we are looking for. This requires that we solve the "coupled" problem of non-linear projection and clustering of data points into those projections simultaneously. Clustering cannot be done in the raw input space because the data really lies in certain non-linear manifolds and the right manifolds cannot be discovered without proper groupings of the data. This is demonstrated in figure 5.1. While most of the work in clustering and projection methods is done independently, attempts have been made to combine them [4, 41]. In this work, we take this "coupling" a step forward by learning clusters and projections simultaneously. This is fundamentally different from an approach like Sparse Subspace Clustering (SSC) [16] that first learns a sparse representation(SR) of the data and then applies spectral clustering to a similarity matrix built from this SR.

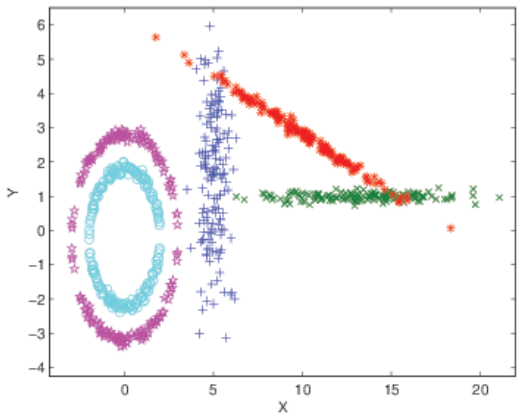


Figure 5.1: Hypothesis 1: Clustering and projection are two coupled paradigms. Clustering cannot be done in the raw feature space because the data lies in latent manifolds. The right manifolds cannot be discovered without clustering the data.

The second hypothesis of this chapter is that in general *further linear clusters might be present in each of the non-linear manifolds*. Thus the overall solution should first find multiple non-linear subspaces within the data and then further cluster the data within each sub-space if necessary. The main contribution of this work is two-fold. A systematic framework for a two-level clustering of input data into meaningful clusters - first level being clustering coupled with non-linear projection while the second level being a linear clustering in each non-linear manifold learnt by first level. We use K-RBMs for the first level clustering and simple k-means on the RBM outputs for the second level clustering. The second contribution of this work is the application of this framework for learning meaningful features from image patches and applying them to improve the image classification accuracy through better features derived from this framework.

Restricted Boltzmann Machines (RBMs) [66] are undirected, energy-based graphical models that learn a non-linear subspace by minimizing reconstruction error. RBMs have gained popularity in recent years because of their wide variety of applications. They have been used successfully to learn

features for image understanding and classification [26], speech representation [46], analyze user rating of movies [60] , and better bag-of-word representation of text data [59]. Moreover, RBMs have been stacked together to learn hierarchical representations such as deep belief networks [26, 7] and convolutional deep belief networks [40] for finding semantically deeper freatures in complex domains such as images. Most nonlinear subspace learning algorithms [28, 55] make various assumptions about the nature of the manifolds they are trying to discover and use a variety of objective functions. RBMs, on the other hand, are a generic framework for learning non-linear subspaces, make no assumptions about the sub-spaces, use a standard energy based learning algorithm, and can model subspaces of any degree of complexity via the number of hidden units making them most suitable as general purpose sub-space learning machines.

Our model learns K RBMs simultaneously. Each RBM represents a subspace manifold in the data. The association of a data point to an RBM depends on the reconstruction error of each RBM for that data point. Each RBM updates its weights based on all the data points associated with it. Through various learning tasks on synthetic and real data, we show the convergence properties, quality of subspaces learnt, and improvement in the accuracies of both descriptive and predictive tasks.

5.2 Training RBMs

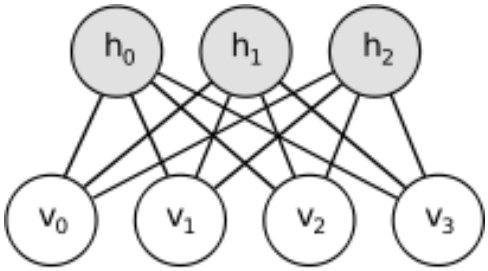


Figure 5.2: A simple Restricted Boltzmann Machine

As described in chapter 2, RBMs are two layered, fully connected networks that have a layer of input/visible variables and a layer of hidden random variables. RBMs model a distribution over visible variables by introducing a set of stochastic features. In applications where RBMs are used for image analysis, the visible units correspond to the pixel values and the hidden units correspond to visual features.

For an RBM with I visible units $v_i, i = 1, \ldots, I$ ($v_0 = 1$ is the bias terms), J hidden units $h_j, j = 1, \ldots, J$ ($h_0 = 1$ is the bias term) and symmetric weighted connections between the visible and hidden layers denoted by $\mathbf{w} \in \mathbb{R}^{(I+1)\times(J+1)}$ (these include asymmetric forward and backward bias terms), the

activation probabilities of units in one layer are computed based on the states of the opposite layer:

$$Pr(h_j|\mathbf{v}) = \sigma\left(\sum_{i=0}^{I} \mathbf{w}_{ij} v_i\right) \tag{5.1}$$

$$Pr(v_i|\mathbf{h}) = \sigma\left(\sum_{j=0}^{J} \mathbf{w}_{ij} h_j\right) \tag{5.2}$$

$\sigma(\cdot)$ is the sigmoid activation function. The RBM energy function, defined as the negative log probability of a configuration of states $(\mathbf{v}, \mathbf{h})$ is given by:

$$-logPr(\mathbf{v}, \mathbf{h}) = E(v, h) = \sum_{i,j} v_i h_j w_{ij} \tag{5.3}$$

Training the RBM thus involves learning the RBM weights and biases that minimize this energy $E(\mathbf{v}, \mathbf{h})$. Most RBM implementations do this in a gradient descent procedure.

Ideally the RBM parameters would be learnt by maximizing the likelihood. This objective function is called the *alternative Gibbs sampling*. However, computing this maximum likelihood involves an exponential number of terms, which makes the training slow and unmanageable. Fortunately, Hinton [25] proposed another objective function called *contrastive divergence* (CD) which is an approximation to the maximum likelihood objective function and can be efficiently minimized. We shall use the CD-1 objective. In the CD-1 forward pass (visible to hidden), we activate the hidden units h_j^+ from visible (input) unit activations v_i^+ (Eq.5.1). In the backward pass (hidden to visible), we recompute visible unit activations v_i^- from h_j^+ (Eq.5.2). Finally we compute the hidden unit activations h_j^- again from v_i^-. The weights are updated using the following rule:

$$\Delta w_{ij} = \eta(< v_i^+ h_j^+ > - < v_i^- h_j^- >) \tag{5.4}$$

where η is the learning rate and $<\cdot>$ is defined as the mean over N examples. The reconstruction error for any sample is computed as:

$$\epsilon = \sum_{i=1}^{I} (v_i^+ - v_i^-)^2 \tag{5.5}$$

There are three kinds of design choices in building an RBM: the objective function used, the frequency of parameter updates, and the type of visible and hidden units. Inspired by most recent methods, we train RBMs by minimizing the contrastive divergence objective (CD-1)[25].

RBM weights are usually updated once per mini-batch. Other options are once per sample update (fully online) and corpus level update (fully batch). We found that doing a full batch update gives a more reliable gradient and slightly better reconstruction compared to mini batch or online updates.

An RBM can have binary or non-binary visible and hidden units. Most RBM implementations use binary visible units. In our applications, we have used Gaussian visible units to model distributions

of real valued data. The stochastic output of hidden unit (Eq.5.1) is always a probability which is thresholded against a random value between 0 and 1 to give a binary activation h_j. In CD-1, it is customary to use binary hidden states when the hidden units are driven by data (h_j^+) and the probabilities without sampling when the hidden units are driven by reconstructions (h_j^-). Thresholding introduces sparsity by creating an information bottleneck. We however always use the activation probabilities in place of their binary states for parameter updates. This process, known as Rao-Blackwellization [9], gives an estimator with lower variance and better clustering performance. This decision was based on the desire to eliminate unnecessary randomness from our approach[1] and was supported by extensive experimentation.

5.3 Learning Multiple Non-Linear Subspaces using K-RBMs

Our non-linear subspace learning model uses K component RBMs. Each component RBM learns a non-linear subspace. The visible units $v_i, i = 1, .., I$ correspond to an I dimensionsional visible (input) space and the hidden units $h_j, j = 1, .., J$ correspond to a learnt non-linear J-dimensional subspace. For the sake of simplicity, we experiment with RBMs of the same size; all the subspaces our model learns have the same true dimensionality J. However, this restriction is unnecessary and we are free to learn subspaces with different true dimensions.

5.3.1 K-RBMs

The K-RBM model has K component RBMs. Each of these maps a set of N sample points $\mathbf{x_n} \in \mathbb{R}^I$ to a projection in $\mathbb{R}^J$. Each component RBM has a set of symmetric weights (and asymmetric biases) $\mathbf{w}^k \in \mathbb{R}^{(I+1)\times(J+1)}$ that learns a non-linear subspace. Note that these weights include the forward and backward *bias* terms. The error of reconstuction for a sample $\mathbf{x}_n$ given by the k^{th} RBM is simply the squared Euclidean distance between the data point $\mathbf{x}_n$ and its reconstruction by the k^{th} RBM, computed using (Eq.5.5). We denote this error by ϵ_{kn}. The total reconstruction error ϵ_t in any iteration t is given by $\sum_{n=1}^{N} \min_k \{\epsilon_{kn}\}$

The K RBMs are trained simultaneously. During the RBM training, we associate data points with RBMs based on how well each component RBM is able to reconstruct the data points. A component RBM is trained only on the training data points associated with it. The component RBMs are given random initial weights $\mathbf{w}^k, k = 1, .., K$.

[1] We use the reconstruction error as a cost function in our clustering; random thresholding introduces randomness in the projections, hence affecting the reconstruction errors.

5.3.2 Clustering using K-RBMs

As in traditional K-means clustering, the algorithm alternates between two steps: (1) Computing association of a data point with a cluster and (2) updating the cluster parameters. In K-RBMs n^{th} data point is associated with k^{th} RBM (cluster) if its reconstruction error from that RBM is lowest compared to other RBMs, i.e. if $e_{kn} < e_{k'n} \forall k \neq k', k, k' \in \{1, .., K\}$.

Once all the points are associated with one of the RBMs the weights of the RBMs are learnt in a batch update. In hard clustering the data points are partitioned into the clusters exhaustively (i.e. each data point must be associated with some cluster) and disjointly (i.e. each data point is associated with only one cluster). In contrast with K-means where the update of the cluster center is a closed form solution given the data association with clusters, in K-RBMs the weights are learnt iteratively.

We can extend our model to incorporate soft clustering where instead of assigning a data point to only one RBM cluster, it can be assigned softly to multiple RBM clusters. The soft association of the n^{th} data point with the k^{th} cluster is computed in terms of the reconstruction error of this data point with the RBM:

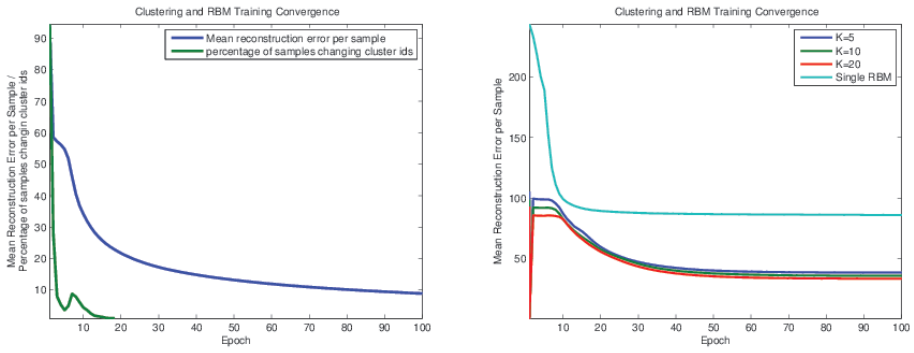
$$\alpha_{nk} = \frac{\exp(-\epsilon_{kn}/T)}{\sum_{k'=1}^{K} exp(-\epsilon_{k'n}/T)} \tag{5.6}$$

where T is the temperature parameter that is reduced over time as in simulated annealing [32]. Each sample $\mathbf{x}_n$ contributes to the training of all RBMs in proportion to its association with the RBMs. While updating weights, the association factor is also multiplied with the learning rate. A K-RBM trained using the soft approach can be seen as a set of RBMs, each of which learns a distribution of all the data but using more information from those it can represent most accurately. Each RBM can reconstruct all the points, some more accurately than the others. This is fundamentally different from the hard clustering where each component RBM learns the distribution of a subset of the data and tries to distort samples from other clusters to look like the samples that it has learnt from.

5.3.3 Initialization and Convergence

Like most EM methods, our model is sensitive to initialization. However, following the standard best RBM implementation practices ensures that this sensitivity is minimal, since the random initial weights are small and the RBM parameters are updated by looking at the data. All our experiments were conducted once with random initialization.

Our clustering framework seeks to learn both the associations (clusters) and the parameters (non-linear subspaces) simultaneously. Thus, here we come across two kinds of convergences: the clustering convergence and the RBM learning (subspace learning) convergence. In our experiments the clustering process is said to have converged when more than 99% of the samples stop changing cluster associations. Usually we require only the cluster associations. We can stop the algorithm once the clustering converges. However, the convergence of clustering just means that the points in each cluster belong to the same non-linear manifold, it does not guarantee the accuracy of the learnt manifolds. If we require



(a) (b)

Figure 5.3: (a) Clustering convergence and RBM training convergence over epochs of the algorithm. Clustering converges long before the RBM reconstruction errors stabilize. (b) A plot of reconstruction errors vs epochs of training process for our experiments on the VOC Pascal dataset in section 5.4.3. Reconstructions are significantly better when we use a K-RBM as opposed to a single RBM. For the Single RBM case, we divide the mean error by 10 to bring it to scale with the others.

data projections in the non-linear subspaces, we continue training the RBMs until the total reconstruction error stabilizes. This is useful, if we intend to further partition the learnt non-linear subspaces (section 5.4.3). In our experiments we found that while the clustering converged within 20 iterations (the data points stopped changing their cluster associations), the reconstruction error continued to drop beyond 100 iterations. We empirically decide the number of epochs our algorithm iterates for and we call this number $maxepoch$. Figure 5.3b shows that K-RBMs significantly outperform the single RBM in terms of the final mean reconstruction error per data point. This clearly validates our first hypothesis that the input data lies in multiple simpler non-linear sub-spaces (multiple K-RBMs) and not in a single complex non-linear sub-space (single RBM).

5.3.4 K-RBMs for Image Feature Learning

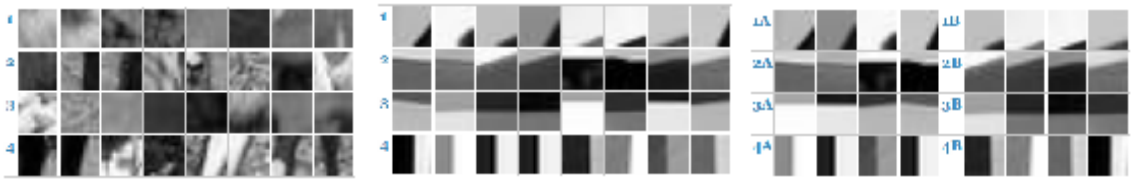
Traditionally, hand-crafted features like SIFT and HoG have been employed for image related tasks. These features are relatively low level and often are not semantically meaningful representations of images. Also they are not learnt but just computed from raw data. Recent times have seen the introduction of features that are learnt from the data. Deep belief networks [40, 46] and convolutional networks [39] have been employed for feature learning to solve a variety of tasks. These methods are based on the hypothesis that good data representations are hierarchical and can be learnt directly from the data; these methods usually have hierarchical layered feature extractors. Although deep learning methods yield robust features, training deep networks involves making many design choices, tuning many parameters, and are often computationally challenging. We propose a feature learning scheme using K-RBMs that learns from the data like the deep networks but is simpler in terms of the overall model complexity and parameters. By doing so, we intend to take a step forward towards promoting feature extraction schemes

that "learn" semantically meaningful representations of the data from the data, while keeping a check on the model complexity.

In image domains, we typically compute local features over patches in an image and then pool the local features to get global image representations (e.g. BoW). Here, we describe dense local K-RBM features. K-RBM features are computed by *hard* clustering patches from dense grids in images. K-RBM features are the projections of these patches in the corresponding learnt manifolds. Unlike the $128-$dimensional SIFT descriptors, the size of the K-RBM features is dictated by the number of hidden units in the component RBMs. In our experiments, we work with patches of size 12×12 pixels. Each patch can thus be represented as a $144-$dimensional sample vector. Our component RBMs have 144 visible units and 36 hidden units. Each local K-RBM feature is thus $36-$dimensional. Unlike SIFT BoW representations where we can perform K-Means clustering of all the SIFT features directly, we can't cluster K-RBM features coming from different component RBMs since they lie in different manifolds. All SIFT features lie in the same $128-$dimensional space. However each K-RBM feature lies in one of K different manifolds. Thus, we cluster the K-RBM features from each component RBM separately, get a different BoW representation for each non-linear manifold and concatenate these BoW representations to get the final BoW representation.

RBMs are generative models that learn a non-linear subspace the data lies in. RBM features are merely projections of the data onto the learnt manifold. The RBM objective minimizes the error of reconstruction of the data from these projections, hence the projections are good "learnt" representations of the data. RBM feature extraction can semantically be understood as non-linear dimensionality reduction of the data. K-RBM feature extraction partitions the data across several RBMs (or manifolds). This has a two-fold advantage: (a) it gives more reliable similarity measures among data in the same manifold, (b) much of the discriminative information is encoded into the data partitions. Figure 5.4 shows image patches corresponding to different BoW/K-RBM clusters for SIFT and K-RBM features. SIFT space is discrete in some sense because it counts the types of edge directions. K-RBMs use a knowledge of the underlying non-linear manifolds to partition the data. In line with our second hypothesis, K-Means followed by K-RBM clustering helps achieve better partitioning of the data and consequently better vector quantization.

Both SIFT and K-RBM project image patches into some non-linear sub-spaces. While SIFT introduces non-linearity by using non-linear filters followed by counting the number of directions the edges take, K-RBMs "learn" features from the data without assuming a specific class of low level features (e.g. edges assumed by SIFT). Thus while SIFT "computes" the features, K-RBMs are more adaptable to the image corpus they are applied to. BoW representations work by counting types of artefacts. While SIFT itself is a histogram of very simple artefacts (edges), K-RBMs treat each patch as an artefact. This fundamental difference can be noticed in figure 5.4 easily.



(a) K-Means on SIFT (b) K-RBM (c) K-Means followed by K-RBM

Figure 5.4: Sample patches corresponding to the different clusters (experiments in section 5.4.4). Each row in (a) and (b) represents a cluster. A row in (c) represents 2 clusters: the concatenation of these 2 clusters gives the cluster in corresponding row in (b). Patches in (a) are independent of (b) and (c). Total number of SIFT clusters in (a) was 1000, K_1 for (b) was 40, K_2 in (c) was 50.

5.4 Applications

5.4.1 Clustering Synthetic Data

In this section, we demonstrate the use of K-RBMs for clustering. We compare the accuracy and speed of K-RBM clustering with the state of the art subspace clustering methods, Random Sample Consensus (RANSAC)[21] and Sparse Subspace Clustering (SSC)[16] in addition to PCA + K-means, t-SNE [71] + K-means and RBM + K-means on two synthetic datasets where we can control the nature of the sub-spaces in the data. t-SNE is a non-linear dimensionality reduction method which minimizes the divergence between distributions over pairs of points. RANSAC works by iteratively sampling a number of points randomly from the data, fitting a model to those points and rejecting outliers. SSC computes a sparse representation (SR) of the data and applies spectral clustering to a matrix obtained from the SR. These algorithms represent *decoupled* learning of projection and clustering.

The goal of these experiments is to investigate the first hypothesis i.e. *clustering* and *projection* are better done in a coupled manner than in a sequential manner. In these experiments, we have compared the performance of a K-RBM with that of KMeans over data processed by a single RBM. In these comparisions, we could either (a) fix the complexity (size) of the latent non-linear subspaces by fixing the number of hidden units in each RBM or (b) fix the number of total RBM parameters in the two models (i.e. if we have a K-RBM with K components having J hidden units each, we allow the single RBM to have KJ hidden units). For our evaluations, we have used the latter scheme, so it is implied that the manifolds learnt by the two models have different dimensionalities. This was done to ensure our model had no undue advantage over the single RBM model which would have a simpler complexity otherwise.

The synthetic datasets in table 5.1 were generated using the code snippet in the SSC demo code (available at http://www.vision.jhu.edu/downloads/). Dataset $D1$ comprises of 500 points drawn from 5 subspaces constructed using orthogonal basis functions, 100 points from each subspace. For all the points, the dimension of the raw feature space is 144 while the true intrinsic dimensionality is 36. $D1$ also contains added Gaussian noise. Dataset $D2$ also consists of 500 points drawn from 5 subspaces.

However, these subspaces differ from the ones in $D1$ in the sense that they don't have orthonormal basis vectors, but oblique projections making it harder than $D1$.

The clustering results are reported in Table 5.1 in terms of misclassification error and mutual information[2] (M.I.) of cluster labels and the known class label. We also quote the running time of these algorithms. We chose 36 principal components for (PCA + K-means). All the RBMs had 144 Gaussian visible units. Each RBM in the K-RBM had 36 binary hidden units while the single RBM had 180. It can be seen that not only K-RBM outperforms these two state of the art methods in terms of quality metrics, it also learns the coupled projection and clustering orders of magnitude faster than these methods both of which are quadratic in the number of data points and hence not very practical to use without serious sampling. Due to the time complexity of RANSAC and SSC it is impractical to train these models on huge datasets without serious sampling.

Table 5.1: Running Time, Misclassification Errors and Mutual Information between cluster and class labels of various methods on synthetic $D1$ and $D2$ datasets.

METHOD	DATASET D1			DATASET D2		
	RUNTIME	ERROR	M.I.	RUNTIME	ERROR	M.I.
K-MEANS	$0.68s$	27.4%	1.9219	$2.76s$	29.6%	1.9219
PCA + K-MEANS	$0.37s$	27.4%	1.9219	$0.42s$	29.8%	1.9219
T-SNE + K-MEANS	$11.68s$	11.3%	1.9619	$11.93s$	23.6%	1.9329
RBM + K-MEANS	$3.29s$	26.6%	1.9219	$3.89s$	28.2%	1.9219
RANSAC	$134.80s$	66.6%	0.1529	$474.72s$	69.6%	0.1499
SSC	$365.29s$	0%	2.3219	$690.93s$	15.6%	2.0732
K-RBM	$0.46s$	0%	2.3219	$3.62s$	0%	2.3219

5.4.2 K-RBMs for clustering MNIST Dataset

In this section, we demonstrate the use of K-RBMs to cluster the MNIST data [39]. The goal of these experiments is to investigate our first hypothesis i.e. *clustering* and *projection* are better done in a coupled manner than in a sequential manner. We compare the clustering performance of K-RBMs with those of K-Means, PCA + K-Means and single RBM + Kmeans. As demonstrated in section 4.1 of the paper, state of the art subspace clustering methods such as RANSAC and SSC are impractical on huge datasets owing to their huge running times. Hence, we refrain from comparing our method with these on MNIST.

In these experiments, we have compared the performance of a K-RBM with that of KMeans over data processed by a single RBM. In these comparisions, we could either (a) fix the complexity (size) of the latent non-linear subspaces by fixing the number of hidden units in each RBM or (b) fix the number

[2]Mutual Information between two random variables X (e.g. cluster labels from a clustering method) and Y (e.g. actual class labels) is defined as: $MI(X,Y) = \sum_{y \in Y} \sum_{x \in X} P(x,y) log(\frac{P(x,y)}{P(x)P(y)})$ where $P(x,y)$ is the joint probability of X and Y and $P(x)$ and $P(y)$ are the marginal probability distributions of X and Y respectively.

of total RBM parameters in the two models (i.e. if we have a K-RBM with k components having J hidden units each, we allow the single RBM to have kJ hidden units). Here, we have used the latter scheme, so it is implied that the manifolds learnt by the two models have different dimensionalities. This was done to ensure our model had no undue advantage over the single RBM model which would have a simpler complexity otherwise.

The MNIST data has $70,000$ data points of binary handwritten digits from 0 to 9. Each data point is size-normalized and centered in a fixed-size (28×28) image represented as a $784-$dimensional binary vector. To test our hypothesis we compare three methods: (i) *PCA + K-means* where the $784-$dimensional raw data is first linearly projected using PCA into a $100-$dimensional subspace and this projected data is then clustered into 10 clusters using standard K-means clustering. (ii) *Single* RBM *+ K-means* where the same $784-$dimensional raw data is first *non-linearly* projected using a single RBM into a *single* $1000-$dimensional subspace (just to keep the complexity compatible) and this projected data is then clustered into 10 clusters using K-means. (iii) K-RBMs where the $784-$dimensional data is clustered using 10 RBMs, each with 100 hidden units. The first two (PCA + K-means and Single RBM + K-means) represent the "decoupled" learning of a projection *followed by* a clustering. The third (K-RBM) represents the *coupled* learning of both a generic non-linear subspace projection *along with* clustering. The K-RBMs are learnt in two modes: with *binary* hidden units (K-RBM-b) and with *real valued* hidden units (K-RBM-r). In both cases the visible units (input data) is binary.

Table 5.2 shows the results comparing the four methods on three (related) metrics: Mutual Information, percentage misclassification error, and weighted mean cluster purity. Both the K-RBM-b and K-RBM-r representing the *coupled* learning of projections and clustering (statistically) significantly outperform the two *decoupled* methods (PCA + K-means and RBM + K-means). Also note that K-RBM-r significantly outperforms K-RBM-b indicating that the loss of information due to binarization of hidden units in pursuit of sparsity may not be the right thing to do in all applications. Figure 4 shows examples of data points in each class (digit) and their cluster centers (obtained by averaging the images of all points associated with cluster), examples of reconstruction of digits from various classes by the single RBM and examples of reconstruction of a digit by all the K-RBM-r here. It can be seen that using linear clustering (K-means) for data that lies in non-linear subspaces is a bad idea. In K-means, the cluster centers may be strikingly different from some of the samples. A single RBM first learns a non-linear subspace that the data lies in and then using K-means on the projections in these subspaces makes sense. However, this is not a very good approach because the data lies in multiple non-linear subspaces. K-RBM clusters the data, learning the subspace of each cluster. These subspaces are specific to each cluster and when samples from other clusters are reconstructed using these subspaces, they are distorted to look like the ones in this cluster.

5.4.3 K-RBMs for Visual Bag-of-Words

These experiments investigate the second hypothesis: multi-variate real-valued data generally lies in multiple non-linear subspace manifolds (e.g. as learnt by K-RBMS) and that there are further potential

Table 5.2: Comparision of coupled vs. de-coupled projection + clustering learning algorithms on MNIST data.

Method	Purity	Error	M.I.
K-means	59.43%	45.23%	1.6651
PCA + K-means	59.36%	45.24%	1.6627
RBM + K-means	60.20%	44.83%	1.6951
k-RBM-b	63.83%	42.79%	1.9127
k-RBM-r	65.16%	38.90%	2.0878

clusters within each of the sub-spaces. This points to a two stage clustering of data: first clustering "coupled" with non-linear projection (e.g. K-RBM) followed by further sub-clustering within each first level cluster. The second goal of these experiments is to propose an alternative to the traditional bag-of-words representations used ubiquitously in computer vision applications.

We experiment with 3 datasets here: PASCAL VOC 2007 [17], 15 Scene Categories [37] and Caltech 101 [19]. PASCAL VOC 2007 data has a total of 5011 training images and 2944 testing images in 20 classes. The 15 Scene Categories dataset has 4485 images in all split over 15 different scene categories. As in [37], we choose 100 random images per category for training and the rest for testing. We repeated the experiments 5 times and report the average accuracy. Caltech 101 has 9146 images, split among 101 distinct object categories. In these experiments, we sampled 30 random images for training from each of the 101 categories, getting a total of 3030 training images; the rest of the images were treated as testing images; however, as in [37], we limited the number of testing images per category to 50. These experiments were repeated 5 times with random subsampling and the mean classification accuracies over the five experiments are reported.

SIFT features on all datasets are computed using a scale of 12 and a shift of 6. Each SIFT feature is 128-dimensional. For the baseline BoW representation, we cluster SIFT features coming from 10 random images per class into 1000 visual words using standard K-means. We use a 2nd level spatial pyramid [37] to get the BoW image representations. For Scene 15 and Caltech 101 datasets, we trained a 1-vs-rest classifier for each class and the test image was assigned the label of the classifier with the highest score. PASCAL VOC 2007 dataset allows multiple classes in the same image; classification accuracy doesn't make sense here. For PASCAL data, we train a 1-vs-rest classifier per class and report the mean Average Precision per class.

In our approach, we create the 1000 clusters in a different way. We first train a K-RBM with K_1 components over SIFT points. The RBMs use $128-$dimensional Gaussan visible units. These are reduced to $20-$dimensional real valued hidden units. Thus the model here is that the feature points in the original 128-dimensional SIFT space reside in K_1 non-linear 20-dimensional subspaces. Once trained, the K-RBM partitions the SIFT data points into K_1 exhaustive and non-overlapping (we used hard clustering) subsets. We further clustered each of the K_1 subsets *in the trasnformed 20-dimensional*

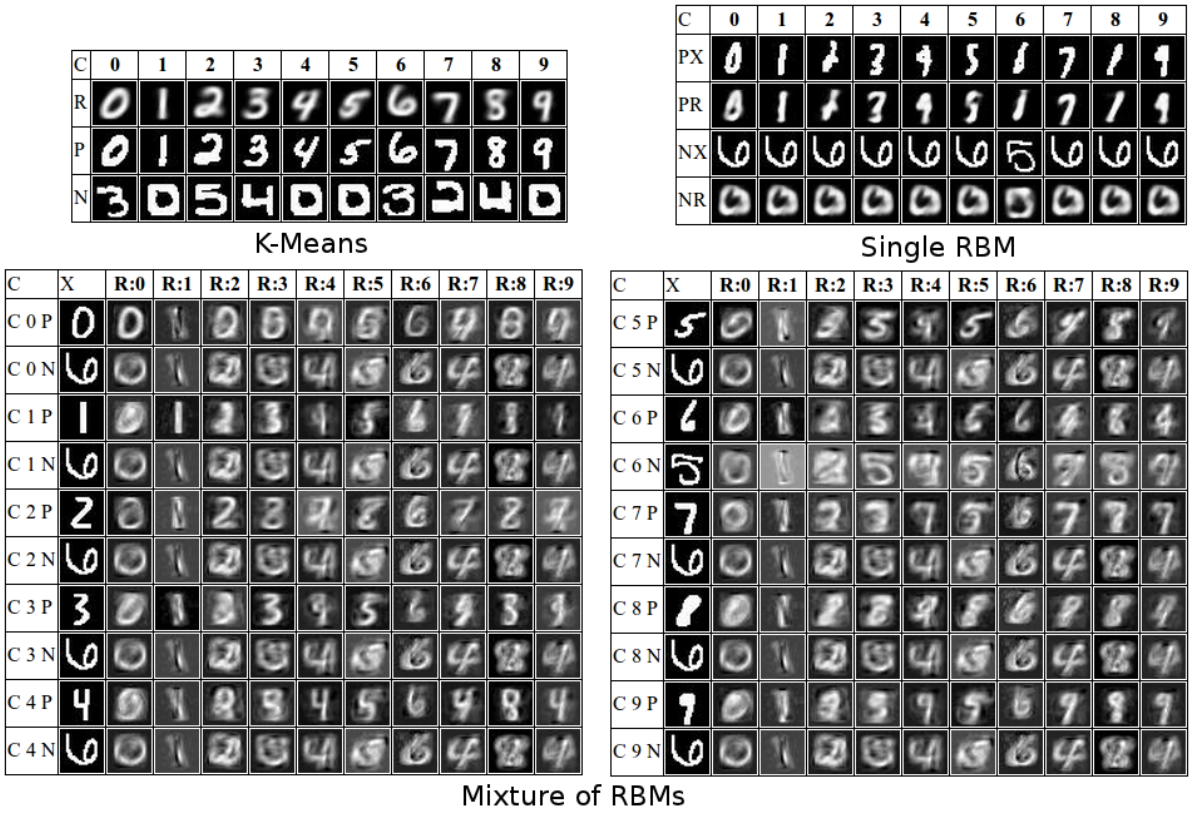


Figure 5.5: Clustering results of K-means, Single RBM + K-means, K-RBM. C denotes the cluster labels, R is the reconstruction in case of RBMs (R:0 is the reconstruction from the RBM corresponding to cluster 0), mean in case of K-means. P is a positive example (correctly classified) from the cluster, N is a negative example (incorrectly classified) from the cluster. X is the data sample.

space into K_2 clusters using simple K-means clustering. This is in-line with our hypothesis that within each sub-space there might be multiple clusters. To keep the total number of clusters compatible with the baseline $K = 1000$, we chose K_1 and K_2 such that their product is 1000. The K_1 and K_2 we report in table 5.4 for different datasets were learnt by using a validation set. Hence, each SIFT descriptor in each image is first mapped to one of the K_1 RBM clusters and then its transformed representation is further mapped to one of the K_2 clusters giving a $K = 1000$ final cluster BoW representation for the images. Here too, we use the 2nd level spatial pyramid for the BoW image representation. The same SVM classifier and evaluation methodology was used for this new image representation.

Overall mean classification average precision (AP) on various code-books on Pascal 2007 is shown in Table 5.3. For K_1 = 8, K_2 = 125, mean AP is highest, significantly higher than traditional BoW. Thus learning clusters in a two-stage process. Learning non-linear manifolds followed by clustering within each manifold improves the quality of the clustering. Also, the right balance has to be struck on how the complexity is distributed between the two stages. The size of projected RBM spaces (in our

case 20-dimensional) is also a factor in the overall complexity of the representation. These need to be empirically determined for any dataset.

The results on the 3 datasets are listed in table 5.4. It can be seen that a 2 level clustering of SIFT features yields better BoW representation. This is indicated by better classification performance on the three datasets. Table 5.4 also lists the mean quantization error of the representations, which is the mean euclidean distance between the SIFT/K-RBM features and the correspoding cluster centers, divided by the length of the feature vector. Note that we normalize the SIFT vectors to contain all values between 0 and 1 (as for K-RBM features) to ensure fair comparision. The quantization errors are significantly smaller for the two way clustering procedure; this indicates better understanding of the feature space.

Table 5.3: Mean Classification AP on VOC Pascal 2007

METHOD	K1	K2	MEAN AP
BASELINE BOW (K-MEANS)	-	1000	52.84%
K-RBM BOW	5	200	55.10%
K-RBM BOW	8	125	56.40%
K-RBM BOW	10	100	55.35%
K-RBM BOW	20	50	54.85%

Table 5.4: Classification Performance on VOC Pascal 2007, 15 Scene Categories and Caltech 101

DATASET	BASELINE BOW		K-RBM BOW	
	PERFORMANCE	MEAN Q.E.	PERFORMANCE	MEAN Q.E.
VOC PASCAL 2007	52.84%	0.7678	$\mathbf{56.40}\%$ ($K_1 = 8, K_2 = 125$)	0.1620
15 Scene	$80.50 \pm 0.5\%$	0.5635	$\mathbf{85.75 \pm 0.6}\%$ ($K_1 = 20, K_2 = 50$)	0.0840
Caltech 101	$68.34 \pm 1.3\%$	0.6420	$\mathbf{72.80 \pm 1.1}\%$ ($K_1 = 8, K_2 = 125$)	0.1365

5.4.4 Feature learning using K-RBMs

In this section, we compare the classification performance of K-RBM features with that of SIFT and Convolutional Deep Belief Networks (CDBN) [40] on Caltech 101 and VOC Pascal 2007 datasets. Note that CDBN classification results are unavailable on VOC 2007. Hierarchical methods such as CDBN work well on Caltech 101 which has object-centered and cropped images, conducive to hierarchical learning of artefacts. Pascal data has huge variation in the scale, position and orientation of objects, even has multiple objects per image. Dense local K-RBM features work well even on Pascal because they exploit the invariance of BoW representations.

SIFT and K-RBM features are computed over a dense grid of 12×12 patches with a shift of 6. The component RBMs have 144 Gaussian visible units and 36 real hidden units. We also use a 2nd level

spatial pyramid [37] to get the BoW Image representations. We fix the BoW vocabulary size to 1000 as in section 5.4. We use a linear pegasos SVM classifier with the χ^2 kernel map for classification [75]. For Caltech 101, as in section 5.4.3, we used 30 random images per class for training and use the rest for testing, limiting the test images to 50 per category. We repeat the experiments 5 times and report the mean classification accuracy. The classification schemes for the two datasets remain the same as in section 5.4.3. K_1, K_2 are learnt using a validation set. The results are reported in tables 5.6 and 5.5. Features learnt using K-RBMs significantly outperform the SIFT and CDBN features. Low level hand-crafted features work well because of scale, distortion invariant pooling schemes like BoW and powerful SVM classifiers. Deep learning methods work because of semantically meaningful features. Our approach combines rich features with powerful BoW representation and SVM classifiers and thus outperforms the two competing classes of methods.

Classification Performance of K-RBM Features on Caltech 101 and VOC Pascal 2007 Datasets.

Table 5.5: Caltech 101

Method	Accuracy
SIFT Features	$68.34 \pm 1.3\%$
CDBN (layers 1+2)	$65.4 \pm 0.5\%$
K-RBM Features ($K_1 = 20$)	$74.2 \pm 1.7\%$

Table 5.6: VOC Pascal 2007

Method	Mean AP
SIFT Features	52.84%
K-RBM Features ($K_1 = 20$)	$\mathbf{58.40}\%$

5.5 Summary

We developed a framework that uses K RBMs to learn rich, complex, and more meaningful features. Compared to the state of the art clustering methods like SSC and RANSAC, K-RBMs is faster and more accurate. The two stage feature learning where first stage uses K-RBMs followed by K-Means for BoW helps improve the overall image representation. K-RBM+K-means features outperform SIFT+Kmeans and CDBN features for image classification. Complex input domains such as images where input lies in multiple non-linear manifolds, the K-RBM approach provides a general, robust, and fast feature learning framework compared to other methods that are either too computationally intensive or make lots of assumptions about the nature of the data or need a lot of parameter tuning. Having said that, it goes without saying that training K-RBMs is slower than training a single RBM, or methods that don't require training at all. This tradeoff between performance and training time needs to be considered carefully when designing large scale systems. So far we have worked with an unsupervised version of K-RBM but this can be extended to supervised version where a separate K-RBM can be learnt for each class.

Chapter 6

Conclusions

In this thesis, we attempt to learn representations for some of the popular computer vision tasks: action recognition, clustering, and visual classification. We study the prominent traditional feature learning approaches in these domains, and devise novel approaches to learn representations for these tasks. We compare our methods with the traditional approaches experimentally and produce evidence to demonstrate the superiority of our methods over the traditional approaches. The major contributions of this thesis are:

PLS kernel for computing similarity between video sequences: In chapter 3, we describe a novel strategy of computing similarity between two video sequences that uses the Partial Least Squares regression. We demonstrate the use of this similarity kernel to solve the tasks of hand gesture recognition on the Cambridge dataset and human activity classification on the UCF sports dataset. Our approach outperforms the previously published state of the art approaches on these two datasets.

Hierarchical Bag of Words using Naive Bayes clustering: In chapter 4, we devise a novel clustering framework for symbolic data. We employ our clustering model to learn hierarchical features starting with BoW symbols, and thus incorporate the traditional BoW model with means to exploit the spatial context in images. It is to be noted that this work is an attempt to bridge the gap between two orthogonal directions in the feature learning community. Our hierarchical feature learning approach combines the advantages of the BoW model and the deep learning methods. We demonstrat the superiority of our representations over the traditional BoW and deep learning approaches by outperforming these on two popular image classification datasets: Caltech 101 and VOC Pascal 2007.

Learning Multiple Non-linear Subspaces using K-RBMs: In chapter 5, we describe a novel approach to non-linear subspace clustering that employs Restricted Boltzmann Machines. We attribute our model's power to the facts that: (a) unlike most popular approaches, it makes no assumptions about the nature of subspaces sought, and (b) it solves the "coupled" problem of learning the subspaces and projections simultaneously. We demonstrate the superiority of our clustering method over traditional

clustering approaches on several synthetic and real datasets (MNIST).

K-RBMs for understanding data: One of our major claims in chapter 5 is that *in general, data is embedded in multiple non-linear subspaces and and within each manifold there may be further clusters.* Driven by this hypothesis, we discuss an overall solution to understanding data that first finds multiple non-linear sub-spaces within the data using K-RBMs and then further clusters the data within each sub-space linearly. We back this claim by using two-level clustering (as opposed to the conventional one level of linear clustering) in the traditional BoW model. We build BoW representations by using K-RBMs for non-linear clustering first and linearly clustering each manifold further. Representations learnt using the two-level clustering outperform traditional BoW on three popular image classification datasets.

K-RBMs for feature learning: We employed the K-RBM clustering algorithm described in chapter 5 for feature learning from raw image patches. We experimentally demonstrate the superiority of K-RBM features over the ubiquitously used SIFT features on popular image classification datasets.

As part of this thesis, we have worked on a variety of computer vision tasks, with a variety of datasets, studied many interesting approaches to learning representations, and finally devised a few strategies of our own that *work*. As mentioned before, "feature learning is not a problem, rather a set of problems, a full fledged field of research per se", and through our efforts, we have attempted to do our bit to contribute to this field. However, research in this field will continue, for solving computer vision is still a distant dream. We conclude this thesis with the final thought: *the location of the holy grail is still obscure and distant, but at times all we can hope is to know that we are walking in the right direction.*

Related Publications

1. **Partial Least Squares Kernel for Computing Similarities between Video Sequences** *(Oral)*
   Siddhartha Chandra & C.V. Jawahar.
   International Conference on Pattern Recognition, Japan, November 2012

2. **Learning Hierarchical Bag of Words using Naive Bayes Clustering**
   Siddhartha Chandra, Shailesh Kumar & C.V. Jawahar.
   Asian Conference on Computer Vision, Korea, November 2012

3. **Learning Multiple Non-Linear Supspaces using K-RBMs**
   Siddhartha Chandra, Shailesh Kumar & C.V. Jawahar.
   IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2013, USA

Bibliography

[1] Wikipedia.

[2] A. Agarwal and B. Triggs. Multilevel image coding with hyperfeatures. In *IJCV*, 2008.

[3] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *SODA*, 2007.

[4] M. S. Baghshah and S. B. Shouraki. Semi-supervised metric learning using pairwise constraints. In *IJCAI*, 2009.

[5] K. Barnard, P. Duygulu, D. Forsyth, N. de Freitas, D. M. Blei, and M. I. Jordan. Matching words and pictures. *JOURNAL OF MACHINE LEARNING RESEARCH*, 3:1107–1135, 2003.

[6] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[7] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, U. D. Montral, and M. Qubec. Greedy layer-wise training of deep networks. In *In NIPS*. MIT Press, 2007.

[8] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2007.

[9] D. Blackwell. Conditional expectation and unbiased sequential estimation. *Ann. Math. Statistics*, 18:105–110, 1947.

[10] O. Boiman, E. Shechtman, and M. Irani. In *CVPR*, 2008.

[11] R. Bowden and M. Sarhadi. Building temporal models for gesture recognition. In *BMVC*, 2000.

[12] M. Bregonzio, J. Li, S. Gong, and T. Xiang. Discriminative topics modelling for action feature selection and recognition. In *BMVC*, 2010.

[13] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2011.

[14] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *ECCV*, 2004.

[15] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *CVPR*, volume 1, 2005.

[16] E. Elhamifar and R. Vidal. Sparse subspace clustering. In *CVPR*, 2009.

[17] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results.

[18] L. Fei-fei. A bayesian hierarchical model for learning natural scene categories. In *In CVPR*, pages 524–531, 2005.

[19] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *WGMBV*, 2004.

[20] R. Fergus, K. Yu, M. A. Ranzato, H. Lee, R. Salakhutdinov, and G. Taylor. Tutorial on deep learning methods for vision. In *CVPR 2012 Tutorial, http://cs.nyu.edu/f̃ergus/tutorials/deep_learning_cvpr12/.*

[21] M. A. Fischler and R. C. Bolles. Random sample consensus. *Commun. ACM*, 1981.

[22] N. Gamage, Y. C. Kuang, R. Akmeliawati, and S. Demidenko. Gaussian process dynamical models for hand gesture interpretation in sign language. In *Pattern Recognition Letters*, 2011.

[23] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *ICCV*, 2005.

[24] K. Grauman and B. Leibe. Synthesis lecture on visual object recognition.

[25] G. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:2002, 2000.

[26] G. E. Hinton, S. Osindero, and Y. whye Teh. A fast learning algorithm for deep belief nets. In *Neural Computation*, 2006.

[27] F. Jurie and B. Triggs. Creating efficient codebooks for visual recognition. In *CVPR*, 2005.

[28] K. Kanatani. Motion segmentation by subspace separation and model selection. In *ICCV*, 2001.

[29] A. V. Ken Chatfield, Victor Lempitsky and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *BMVC*, 2011.

[30] T.-K. Kim and R. Cipolla. Gesture recognition under small sample size. In *ACCV (1)*, 2007.

[31] T.-K. Kim, S.-F. Wong, and R. Cipolla. Tensor canonical correlation analysis for action classification. In *CVPR*, 2007.

[32] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 1983.

[33] A. Kovashka and K. Grauman. Learning a hierarchy of discriminative space-time neighborhood features for human action recognition. In *CVPR*, 2010.

[34] Y. lan Boreau, F. Bach, Y. Lecun, and J. Ponce. Learning mid-level features for recognition. 2010.

[35] D. Larlus and F. Jurie. Latent mixture vocabularies for object categorization. In *BMVC*, 2006.

[36] S. Lazebnik, C. Schmid, and J. Ponce. A maximum entropy framework for part-based texture and object recognition. In *ICCV*, 2005.

[37] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, pages 2169–2178, 2006.

[38] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[39] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

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

[41] G. Liu, Z. Lin, and Y. Yu. Robust subspace segmentation by low-rank representation. In *ICDMW*, 2010.

[42] J. Liu and M. Shah. Scene Modeling Using Co-Clustering. In *ICCV*, 2007.

[43] D. G. Lowe. Distinctive image features from scale-invariant keypoints. In *IJCV*, 2003.

[44] Y. M. Lui and J. R. Beveridge. Tangent bundle for human action recognition. In *FG*, 2011.

[45] Y. M. Lui, J. R. Beveridge, and M. Kirby. Action classification on product manifolds. In *CVPR*, 2010.

[46] A. Mohamed, G. Dahl, and G. Hinton. Deep belief networks for phone recognition. In *ICASSP*, 2011.

[47] F. Moosmann, E. Nowak, and F. Jurie. Randomized clustering forests for image classification. *PAMI*, 2008.

[48] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *CVPR*, 2006.

[49] A. Oliva and A. Torralba. Building the gist of a scene: the role of global image features in recognition. In *Progress in Brain Research*, page 2006, 2006.

[50] V. Pavlovic, R. Sharma, and T. Huang. Visual interpretation of hand gestures for humancomputer interaction: A review. In *IEEE Trans. Patt. Anal. Mach. Intell., Vol. 19*, 1997.

[51] F. Perronnin. Universal and Adapted Vocabularies for Generic Visual Categorization. 2008.

[52] F. Perronnin, J. Snchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *IN: ECCV*, 2010.

[53] T. Quack, V. Ferrari, B. Leibe, and L. V. Gool. Efficient mining of frequent and distinctive feature configurations. In *ICCV*, 2007.

[54] P. Quelhas, F. Monay, J. M. Odobez, G. D. Perez, and T. Tuytelaars. A Thousand Words in a Scene. *PAMI*, 2007.

[55] S. Rao, R. Tron, R. Vidal, and Y. Ma. Motion segmentation via robust subspace separation in the presence of outlying, incomplete, or corrupted trajectories. In *CVPR*, 2008.

[56] M. Riesenhuber, T. Poggio, and E. Studies. Hierarchical models of object recognition in cortex, 1999.

[57] M. D. Rodriguez, J. Ahmed, and M. Shah. Action mach: a spatio-temporal maximum average correlation height filter for action recognition. In *CVPR*, 2008.

[58] R. Rosipal and N. Kramer. Overview and recent advances in partial least squares. In *Lecture Notes in Computer Science*, 2006.

[59] R. Salakhutdinov and G. Hinton. Replicated softmax: an undirected topic model. In *In NIPS*, 2010.

[60] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML*, 2004.

[61] D. Schonfeld. Motionsearch: Context-based video retrieval and activity recognition in video surveillance. In *AVSS*, 2009.

[62] A. Shamaie and A. Sutherland. Graph-based matching of occluded hand gestures. In *Proc. of the Applied Imagery Pattern Recognition*, 2001.

[63] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[64] J. Sivic, B. Russell, A. Efros, A. Zisserman, and W. Freeman. Discovering objects and their location in images. In *ICCV*, 2005.

[65] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, 2003.

[66] P. Smolensky. In *Parallel Distributed Processing: Volume 1: Foundations*. 1987.

[67] M. Su, H. Huang, C. Lin, and C. Huang. Application of neural networks in spatio temporal hand gesture recognition. In *Proc. of the IEEE World Congress on Computational Intelligence*, 1998.

[68] S. K. D. Tomasz Malisiewicz. Learning visual subcategories for basic-level categorization. In *CVPR, Fine-Grained Visual Categorization Workshop*, 2011.

[69] T. Tuytelaars and C. Schmid. Vector quantizing feature space with a regular lattice. In *ICCV*, 2007.

[70] S. Ullman, M. Vidal-Naquet, and E. Sali. Visual features of intermediate complexity and their use in classification. *Nature neuroscience*, pages 682–687, July 2002.

[71] L. van der Maaten and G. Hinton. Visualizing Data using t-SNE. In *JMLR*, 2008.

[72] J. C. van Gemert, J.-M. Geusebroek, C. J. Veenman, and A. W. M. Smeulders. Kernel codebooks for scene categorization. In *ECCV 2008, PART III. LNCS*, pages 696–709. Springer, 2008.

[73] J. C. van Gemert, J.-M. Geusebroek, C. J. Veenman, C. G. M. Snoek, and A. W. M. Smeulders. Robust scene categorization by learning image statistics in context. In *SLAM - CVPR*, 2006.

[74] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms, 2008.

[75] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. In *CVPR*, 2010.

[76] J. Vogel and B. Schiele. Natural scene retrieval based on a semantic modeling step. In *In CIVR*, 2004.

[77] J. Vogel and B. Schiele. Semantic modeling of natural scenes for content-based image retrieval. *IJCV*, 2007.

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

[79] A. D. Wilson and A. Bobick. Parametric hidden markov models for gesture recognition. In *IEEE Trans. Patt. Anal. Mach. Intell*, 1999.

[80] J. Winn, A. Criminisi, and T. Minka. Object categorization by learned universal visual dictionary. In *ICCV*, 2005.

[81] H. Wold. Path models with latent variables: The NIPALS approach. In *Quantitative Sociology: International perspectives on mathematical and statistical model building*, 1975.

[82] J. Yang, K. Yu, Y. Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *CVPR*, 2009.

[83] L. Yang, R. Jin, C. Pantofaru, and R. Sukthankar. Discriminative cluster refinement: Improving object category recognition given limited training data. In *CVPR*, 2007.

[84] X. Zhou, K. Yu, T. Zhang, and T. S. Huang. Image classification using super-vector coding of local image descriptors. In *ECCV*, 2010.