ABSTRACT

Title of dissertation: **RICH AND EFFICIENT VISUAL DATA REPRESENTATION**

Mohammad Rastegari, Doctor of Philosophy, 2016

Dissertation directed by: **Prof. Larry S. Davis**

Increasing the size of training data in many computer vision tasks has shown to be very effective. Using large scale image datasets (e.g. ImageNet) with simple learning techniques (e.g. linear classifiers) one can achieve state-of-the-art performance in object recognition compared to sophisticated learning techniques on smaller image sets. Semantic search on visual data has become very popular. There are billions of images on the internet and the number is increasing every day. Dealing with large scale image sets is intense per se. They take a significant amount of memory that makes it impossible to process the images with complex algorithms on single CPU machines. Finding an efficient image representation can be a key to attack this problem. A representation being efficient is not enough for image understanding. It should be comprehensive and rich in carrying semantic information. In this proposal we develop an approach to computing binary codes that provide a rich and efficient image representation. We demonstrate several tasks in which binary features can be very effective. We show how binary features can speed up large scale image classification. We present learning techniques to learn the binary features from supervised image set (With different types of semantic supervision; class labels, textual descriptions). We propose several problems that are very important in finding and using efficient image representation.

RICH AND EFFICIENT VISUAL DATA REPRESENTATION

by

Mohammad RASTEGARI

Dissertation submitted to the Faculty of the Graduate School of the University of Maryland, College Park, in partial fulfillment of the requirements for the degree of Doctor of Philosophy

2016

"When David Marr at MIT moved into computer vision, he generated a lot of excitement, but he hit up against the problem of knowledge representation; he had no good representations for knowledge in his vision systems."

Marvin Minsky

Acknowledgements

I am grateful for having an advisor (Prof. Larry Davis) who let me to have mistakes to create a road toward success. Who taught me research is not about stacking scientific knowledge in a single mind, but about conveying the knowledge to others mind. I am also very thankful to Dr. Ali Farhadi for his support through the difficulties during the early years of my graduate study. I thank to other faculty (Prof. David Forsyth, Prof. David Jacobs and Prof. Hal DaumeIII) who helped me during my research. I am thankful to my colleagues (M. Najibi, J. Choi, S. Fakhraei, C. Fang and F. Mirrashed) without whom I could not accomplish any of the works.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the past decade the usage of large databases for object and speech recognition have shown that simple learning methods such as linear classifiers and K-Nearest Neighbors can result in comparable performance to complex learning methods. This has led to widespread discussion about how much “Data Matters”, which posits that having more data is more effective than having a complex model. Resolving the extent to which “data matters” is confounded by computational and algorithmic challenges in learning a complex model on a large-scale dataset.

On the one hand, the problem of learning complex models on big data has been addressed by investigating the use of high-performance and parallel computing. A complementary approach to dealing with big data is developing efficient data representations which enable efficient processing. Representing data using compact binary codes is one of the most powerful ways to achieve efficient data representation. It has received significant attention during the past years.

Binary codes are attractive image representations for image search and retrieval because they are easy to match, and the capacity of the space of even very short binary codes is so large that all of the digital images in the world can be indexed with relatively short binary codes. 64-dimensional codes can index about 10^{19} images - 5 times the estimated number of bits created in 2002 and likely similar to the number of digital images in existence. Unfortunately, it is not known how to perfectly encode visual information into binary codes to enable efficient search and retrieval. So far, encoding visual or textual information into binary codes has been based on preserving some notion of distance or similarity in the original visual feature space. Most of the binary code construction methods encode images so that the Hamming distances between similar looking images are smaller than for dissimilar images. But it is obvious that images that share the same semantics (e.g. same category of objects, scene, action, ...) can have very different visual appearance. Therefore, if binary encoding is to be more generally useful for recognition and image understanding, we require an encoding method that can capture the semantics of the data. However, finding such an encoding method might be as difficult as the recognition task itself. It depends on what level of semantics we want to represent with the binary codes. For instance, in categorization, we might want to encode each image by a binary number that indicates the category number assigned to that image. But this is equivalent to solving the categorization task, which is difficult.

1.1 Main Objective

The main objective in this PhD is finding efficient representations of data (so far compact binary codes) that A) capture enough semantics to enable accurate recognition and understanding in a variety of tasks in machine learning. B) can be extracted efficiently from the original feature space (in my research mostly visual and textual space) to make it practical to be applied to large-scale data and complex models. Finding such a representation should lead to significant progress in a wide range of machine learning problems. I am also interested in exploiting these binary codes for discovering efficient learning algorithms. In pursuing these goals, I have completed several projects on image and text understanding. Some of them are published in prestigious conferences and others are submitted or still in progress. We showed how a certain class of binary codes can efficiently and accurately solve a variety of problems in computer vision. We introduced the notion of predictability of code bits, which enables each bit of the binary code to be reliably predicted from the original feature space. We showed that predictability plays an important role in recognition problems. In recent work, we showed that domain adaptation can significantly benefit from predictable binary codes and give significant improvement over state-of-the-art techniques. We elucidated a close relation between predictable code bits and visual attributes. My research has shown how these predictable codes can be effectively applied to a wide variety of tasks (Patch based image restoration, cross modality hashing, video clustering, biometric recognition, etc.). This is especially true in the case of high dimensional data where even operations like hashing become expensive because of costly projection operators. Unlike most hashing methods that sacrifice accuracy for speed, we propose a novel method that improves the speed of high dimensional image retrieval by several orders of magnitude without any significant drop in performance. In the remainder of this thesis I will describe what have been done so far and the main problems that I am interested in addressing.

1.2 Binary Coding

I began my research on binary codes for efficient object recognition. We assumed that we are given binary codes for images and wanted to address the problem of object-class retrieval in large image data sets: given a small set of training examples defining a visual category, the objective is to efficiently retrieve images of the same class from a large database (Tens of millions of images) using a PC with a single CPU. We proposed two contrasting retrieval schemes achieving good accuracy and high efficiency. We introduced a novel ranking procedure that provides a significant speedup over inverted file indexing when the goal is restricted to finding the top-k (i.e. the k-highest ranked) images in the data set. My algorithms for object-class retrieval can search a 10 million item database in just a couple of seconds on a PC and produced categorization accuracy comparable to the best known class-recognition systems. The details are presented in 2 The result of this work was published in IEEE International Conference on Computer Vision (ICCV) 2011 [2].

In contrast to conventional binary embedding methods, we were looking for a binary embedding that discriminates between object categories and at the same time can be well predicted from visual data. The most discriminative codes (like assigning unique codes to examples from the same category) are extremely hard to predict from visual data. And the most predictable codes may contain very little information about categories, therefore resulting in poor discrimination. A code is discriminative, if examples of different categories appear far away from each other and instances of the same category lie close by. Each bit of a code is generated by checking on which side of a hyperplane an instance lies. We proposed that a bit is predictable (can be reliably predicted from visual data), if the corresponding hyperplane for generating that bit has a large margin from data points. We proposed a model that balances between discrimination and predictability of the codes. Each bit in that model corresponds to a meaningful characteristic of an image - it can be thought as a visual attribute. By looking at images on both sides of a hyperplane corresponding to a bit, we could clearly see that images on each side often shared a common visual characteristic (e.g. rounded shape objects, animals, tall buildings, etc.). Using these binary codes as image features, we achieved state-of-the-art classification accuracy on very challenging datasets (Caltech256, ImageNet). Chapter 3 gives a comprehensive explanation of this work.This work was published in the European Conference on Computer Vision (ECCV) 2012 [3].

These discriminative binary codes turned out to be very useful for other tasks. Collecting large and effective training data is crucial for building accurate visual models, but expensive. We designed a method exploiting these binary codes to expand the visual coverage of training sets that consist of a small number of labeled examples. This method adds images to a category from a large unlabeled image pool. We showed significant improvement in category recognition accuracy evaluated on a large-scale dataset, ImageNet. The result of this work was published in the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2013 [4].The details of this work are presented in Chapter 5

Predictability of each bit is a crucial factor for a binary code to be a robust feature. This encouraged me to use the notion of predictability in Domain Adaptation, where training (source) and testing (target) data are drawn from different distributions (domains). We proposed an iterative algorithm that learns predictable binary codes in the target domain and uses them as features in the source domain. This simple method showed very impressive results on several challenging datasets. A paper describing this work appeared in the IEEE International Conference on Computer Vision (ICCV) 2013 [5].Chapter 7 provides the details.

Very often, class labels of an image serve as a semantic representation. But semantics can also be conveyed through sentences paired with an image, which describe image contents. Typically, we have paired data from two different modalities or views (e.g. Visual, Textual). This raises the interesting problem of how to embed the data from two different views into a shared binary space. Such a shared binary space would enable very accurate and efficient cross-modality search. For example, given an image, we can generate a binary code that is very similar to the binary code for the sentence describing that image and vice versa (sentence-to-image). We took advantage of the predictability of bits and designed a dual-view hashing algorithm. Briefly, we used the binary values of each bit in the first view as a label for learning a max-margin hyperplane in the second view (and vice-versa). We thus learned shared predictable binary codes across the two views. A paper on this project appeared in the International Conference on Machine Learning (ICML) 2013 [6].Chapter 4 elaborates the details of this work.

Another application of these binary codes is image search when the query consists of a number of words. We refer to this as multi-attribute query. Users often have very specific visual content in mind that they are searching for. The most natural way to communicate this content to an image search engine is to use keywords that specify various properties or attributes of the content. In a multi-attribute image search, some combinations of attributes can be learned jointly, resulting in a better classifier. For example, when looking for “white, furry, dog” it might be better to train a white-furry classifier and fuse it with a dog classifier. Detecting which conjunction of the attributes to be selected is very expensive. A naive approach is to learn classifiers for all possible combinations and evaluate them on a validation set and pick the combination that gives the best validation accuracy. But we want to do this all at query time. It is not possible to do them offline because we do not know what attributes will be requested by user at query time. We exploit the binary space and developed an efficient optimization that gives a score to each combination of attributes without learning any classifier at query time. We use the intuition that geometric notions that capture the compactness (intra- class variance) of the set of images that satisfy a combination (e.g. white-dog), and the margin of these images from other distracter images (inter-class variance) provide good proxies for the likely effectiveness of a classifier trained to recognize the combination. We showed that these geometric quantities could be evaluated efficiently in a discriminative binary space. We published a paper out of this work at IEEE International Conference on Computer Vision and Pattern Recognition (CVPR) 2013 [7]. Chapter 6 presents the general idea and details of this work.

In Chapter 8 we point to very important criteria on evaluation of binary coding methods. We found in many recent papers that when they compare results from different binary coding, they are not conducting a fair comparison. The common misunderstanding relates to the goal of binary code learning methods. Some binary coding methods optimize codes to preserve Euclidean distance in the original feature space while others optimize to preserve angular distance. Many papers directly compare the output of these methods, but they are not directly comparable. In chapter 8 we propose appropriate evaluation methodologies for these methods; our comparative experimental results reveal different performance previously reported methods.

In Chapter 9 we proposed a fast technique for binary encoding that can be used for efficient high dimensional image retrieval tasks. This method uses a fixed computational budget to learn a projection matrix that generates binary codes. To do this, we propose to learn computationally bounded sparse projections for the encoding step. To further increase the accuracy of the method, we add an orthogonality constraint on projections to reduce bit correlation. We then introduce an iterative scheme that jointly optimizes this objective, which helps us obtain fast and efficient projections. We demonstrate this technique on large retrieval databases, specifically ImageNET, GIST1M and SUN-attribute for the task of nearest neighbor retrieval, and show that our method achieves a speed-up of up to a factor of 100 over state-of-the-art methods, while having on-par and in some cases even better accuracy.

In Chapter 10 we focused on the problem of large scale retrieval using ANN. A new general approach for multi-dimensional n-ary coding -Linear Subspace Quantization (LSQ)- was introduced for ANN. LSQ achieves lower reconstruction error than other n-ary coding methods. Furthermore, it preserve the similarities in the original space, which is important when it is used directly for learning tasks. Experiments show that LSQ outperforms other binary and n-ary coding methods on large scale image retrieval. We also compared the performance of binary and n-ary coding methods for this task. We showed that n-ary coding outperforms binary coding when distance estimation is used to reduce the search computation cost. However, in combination with Subset Indexing, interestingly, binary coding works better for retrieval.

Chapter 2

Binary Codes for Large Scale Object-Class Retrieval

2.1 Overview

Over the last decade the accuracy of object categorization systems has dramatically improved thanks to the development of sophisticated low-level features [8–10] and to the design of powerful nonlinear classification models that can effectively combine multiple complementary descriptors [11]. However, these categorization systems do not scale well to recognition in large image collections due to their large computational costs and high storage requirements.

In a parallel line of research, the efficiency of image retrieval systems has rapidly progressed to the point of enabling real-time search in millions of images [12]. Scalability to large data sets is typically achieved by casting image search as a text retrieval problem. The analogy with text-retrieval is made possible by representing each image as a sparse histogram of quantized features, known as visual words [13]. However, this representation is best suited to implement low-level notions of visual similarity. As a result, these systems are primarily used to detect near duplicates [14] or to find images containing the same *object instance* as the one present in a given query photo [15, 16]. Hashing methods have been used to compute low-dimensional image signatures encoding the overall global structure present in an image [17]. While such representations can be used to efficiently find photos matching the global layout of a query image, they have not been shown to be able to produce good categorization accuracy.

The objective of this work is to bridge these two independent lines of research. We borrow data structures and methods from image retrieval – such as sparse and compact descriptors, inverted files, as well as algorithms for approximate distance calculation – to develop a system for accurate and efficient *object-class* search in large image collections: given a training set of examples defining an arbitrary object class (not know before query-time), our algorithms can efficiently find images of this category in a large database. We envision such system to be used as a tool to interactively search in collections of unlabeled images, such as community-provided pictures or albums of personal photos.

Our approach uses the binary "classeme" descriptor of Torresani et al. [18] as representation for images. The binary entries in this descriptor are the Boolean outputs of a set of nonlinear object classifiers evaluated on the image. These base classifiers are trained on an independent data set obtained using text-based image search engines. Intuitively, the classeme descriptor provides a high-level description of the image in terms of similarity to the set of base classes. The approach is analogous in spirit to image representations based on attributes [19–21], which are human-defined properties correlated to the classes to be recognized. As classeme vectors provide a rich semantic description, they have been shown to produce good categorization accuracy even with simple linear classification models, which are efficient to evaluate. Furthermore, these descriptors are compact in size (the dimensionality of the binary vector is 2625, corresponding to only 329 bytes/image) and thus allow storage of large databases in memory for efficient recognition.

In this work we investigate the benefits of using sparse classification models with attribute features, i.e. classifiers that are explicitly constrained to use only a small set of attributes for the recognition of a new query category. The sparsity of the models enables the advantageous use of inverted files [22] for efficient retrieval. Furthermore, we present a new ranking algorithm which exploits the sparsity of the classifier to find the top-k images with even lower computational cost. We show that this simple approach produces a 24-fold speedup over brute-force evaluation.

We compare these sparsity-based retrieval models with a system that uses vector quantization to efficiently approximate the ranking scores of images in the database. We demonstrate that at the cost of a small drop in accuracy, this algorithm achieves similar efficiency and superior scalability in terms of memory usage.

2.2 Background

As already pointed out above, sparse retrieval models have been extensively used in image search [13, 15, 16]. Among the methods in this genre, the min-hash technique of Chum et al. [14] bears a close resemblance to our approach. It measures similarity between two images in the form of an approximate weighted histogram intersection. This measure can be efficiently calculated using inverted lists over sketch hashes, which are tuples of randomly-selected feature subsets. However, the weights of the histogram intersection must be defined a priori, before the creation of the index, and therefore this approach can only implement static similarity measures. Our task instead requires tuning the similarity measure for every new query category. The approach in [23] also uses inverted lists defined over feature subsets, called visual phrases. The visual phrases are binary "same-class" classifiers trained to recognize whether two given images belong to the same class or to different classes, regardless of the category. In principle this model could be adapted to be used for class retrieval. However, the visual phrases output only a Boolean value and thus cannot provide a ranking.

The idea of training sparse classification models on attribute vectors is similar in spirit to the "Object Bank" approach of Li et al. [24]. The Object Bank is a high-level image representation encoding the spatial responses of a large set of object detectors applied to the image. However, the dimensionality of these representation is too high to allow storage in memory for large collections (each descriptor contains 44,604 real values), which is the problem considered here. Li et al. have shown that the Object Bank descriptor can be compressed down to compact sizes using regularization terms enforcing feature sparsity. However, the feature selection is optimized with respect to a fixed set of classes, and the generalization performance of the selected features has not been demonstrated on novel classes. The focus of our work instead is on the design of compact models and efficient methods that can support search of arbitrary novel classes.

Our algorithm for approximate ranking shares similarities with efficient techniques for approximate nearest neighbor search. Most of these methods operate by embedding the original feature vectors into a low-dimensional space via hashing functions or nonlinear projections such that the Hamming distance or the L2 distance in this space approximates a given metric distance in the original high-dimensional space [25–27]. In particular, Jain et al. [28] have proposed an algorithm that learns a Mahalanobis distance from similarity constraints and encodes this learned distance metric into randomized locality-sensitive hash functions. The resulting system enables efficient and accurate class recognition in large data sets. The work of Kulis and Grauman [29] has extended this approach to generate locality-sensitive hashing functions that can approximate arbitrary kernel distances. However, these methods optimize the categorization metric and the embedding space with respect to a fixed set of classes during an offline training stage, *before* the search. Instead our problem statement requires to efficiently learn *at query time* the classification model for a category that is not known in advance.

2.3 Object-Class Search

We now formally define our problem statement and introduce the notation that will be used in the rest of the paper. We assume we are given a database of N unlabeled images, from which binary vectors are extracted during an offline stage. We indicate with $\mathbf{x}_i \in \{0,1\}^D$ the binary classeme descriptor computed from the i-th image in the database. These feature vectors are stored in an index for subsequent efficient retrieval. At query time we are given a set of n^+ training images belonging to an arbitrary query category as well as n^- negative examples. In a practical application the negative set could be a fixed "background" collection containing examples of many different categories (we consider such scenario in one of the experiments). We indicate with $\tilde{\mathcal{D}}$ the labeled training set obtained by merging these two sets, i.e., $\tilde{\mathcal{D}} = \{(\tilde{\mathbf{x}}_1, \tilde{y}_1), \ldots, (\tilde{\mathbf{x}}_n, \tilde{y}_n)\}$ where $\tilde{\mathbf{x}}_i$ is the binary classeme vector of the i-th training example, $\tilde{y}_i \in \{0, 1\}$ indicates its binary label and $n = n^+ + n^-$ (note that we use the ˜ symbol to differentiate training example $\tilde{\mathbf{x}}_i$ from database example $\mathbf{x}_i$). The objective of the system is to efficiently retrieve relevant database images and rank them according to the probability of belonging to the query category. We evaluate the accuracy of the retrieval system in terms of precision at k. Again, we want to emphasize that the object classes provided at query time are not known in advance. Thus, the system must be able to learn the retrieval model exclusively from the set $\tilde{\mathcal{D}}$.

Since our objective is to retrieve images belonging to the query class, we employ binary classifiers as retrieval models and use their classification output as ranking score. We restrict our study to linear classifiers, since they are efficient to learn as well as to evaluate. Thus, the ranking score for database example $\mathbf{x}$ is computed as $h_\theta(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$, where $\boldsymbol{\theta}$ is a D-dimensional vector of parameters (we incorporate a bias term in the weight vector by adding a constant entry set to 1 for all examples). Our classifiers are learned by optimizing an objective function of the form:

$$E(\boldsymbol{\theta}) = R(\boldsymbol{\theta}) + \frac{C}{m} \sum_{i=1}^{m} L(\boldsymbol{\theta}; \tilde{\mathbf{x}}_i, \tilde{y}_i) \tag{2.1}$$

where R is a regularization function aimed at preventing overfitting, L is the loss function penalizing misclassification, and C is a hyper-parameter trading off the two terms.

2.4 Efficient category retrieval via sparsity

Retrieval efficiency can be achieved by forcing the classifier h_θ to be sparse, i.e., by requiring the parameter vector $\boldsymbol{\theta}$ to have very few non-zero entries so that the evaluation cost will be a small fraction of D per image. To realize this efficiency we can use an *inverted file* with D entries, each providing the list of database images containing one of the D binary features.

2.4.1 Sparse classifiers

The literature on sparse linear classifiers is vast and review of these methods is beyond the scope of this paper. In our work, we restrict our attention to the following sparse classification models due to their good balance of feature sparsity and accuracy:

- **L1-LR:** this is a ℓ_1-regularized logistic regression classifier [30] obtained by defining $R(\boldsymbol{\theta}) = ||\boldsymbol{\theta}||_1$ and $L(\boldsymbol{\theta}; \tilde{\mathbf{x}}_i, \tilde{y}_i) = \log(1+\exp(-\tilde{y}_i\boldsymbol{\theta}\cdot\tilde{\mathbf{x}}_i))$. The ℓ_1-regularization is known to produce sparser parameter vectors than the more conventional ℓ_2-regularization.

- **FGM:** this is the Feature Generating Machine described in Tan et al. [31]. It minimizes a convex relaxation of the constrained optimization obtained by setting $R(\boldsymbol{\theta}) = ||\boldsymbol{\theta}||_2$, $L(\boldsymbol{\theta}, \mathbf{d}; \tilde{\mathbf{x}}_i, \tilde{y}_i) = \max(0, 1 - \tilde{y}_i(\boldsymbol{\theta} \odot \mathbf{d}) \cdot \tilde{\mathbf{x}}_i)^2$ and by enforcing constraint $\mathbf{1} \cdot \mathbf{d} \leq B$ where $\odot$ denotes the elementwise product between vectors, $\mathbf{d} \in \{0,1\}^D$ is a binary vector indicating the active features, and B is a hyperparameter controlling the number of nonzero weights. A cutting plane algorithm is employed to efficiently find the sparse features defined by vector $\mathbf{d}$. This method has been shown to produce state-of-the-art results in terms of sparsity and generalization performance.

We contrast these sparse classifiers with a traditional linear SVM using an L2 regularization term. We denote this classifier with **L2-SVM**.

2.4.2 Top-k pruning

In this subsection we present an algorithm that exploits sparsity to efficiently find the top-scoring k images in the database using the linear retrieval functions described above. This approach is well suited to our intended retrieval application since a user is typically interested in only the top search results. The key-idea of this ranking algorithm is to update lower and upper bounds on the scores of the images to gradually prune the candidate set without complete calculation of the classification outputs. An upper bound $u(i)$ and a lower bound $l(i)$ is defined for every image i in the database. The upper bound $u(i)$ is first initialized to score u^*, which is the highest possible score achievable given the weight vector $\boldsymbol{\theta}$. Such score is obtained when a binary feature vector contains nonzero values precisely in the positions where the weight vector $\boldsymbol{\theta}$ has positive values and it contains 0 in the entry positions where the weights are negative. Analogously, $l(i)$ is initialized to the lowest possible score l^*, which occurs when the entries of the feature vector are 0 in positions where the weights are positive and 1 where the weights are negative. Then, these bounds are updated by considering one weight entry at a time. Let p_d be the entry considered in the d-th iteration. Let us assume that $\theta(p_d)$ is positive (the case when this value is negative is analogous). Then, if the binary vector of the i-th image contains a 1 in position p_d, the lower bound $l(i)$ will be incremented by $\theta(p_d)$; if instead $x_i(p_d)$ is 0, the upper bound $u(i)$ will be decremented by $\theta(p_d)$. Thus, we see that at each iteration d the gap between the lower and the upper bound for each image i is decreased by amount $\theta(p_d)$. In order to produce the fastest reduction of this gap, we process the weights in descending order of absolute values. Furthermore, for efficiency in our implementation we only store and update the lower bound for each image, since the upper bound is trivially obtained by adding an iteration-dependent value which is constant for all images. This derives trivially from observing that at any iteration d the gap between the bounds for every image i is: $u(i) - l(i) = u^* - l^* - \sum_{d'=1}^{d} |\theta(d')|$. At each iteration, after updating the bounds, the algorithm identifies the set A of k images having highest lower bounds (this can be done in a linear scan over the vector l). Then, in the pruning step, the method eliminates from further consideration the images having upper bound smaller than the minimum lower bound in the set A, since such images cannot rank in the top-k. The pruning rate will obviously depend on the distribution of the weights in the vector $\boldsymbol{\theta}$ and the statistics of classemes. Intuitively, the pruning rate will be high when $\boldsymbol{\theta}$ is sparse and when the weight magnitudes decay rapidly when sorted in decreasing order. Indeed in the experiment section we empirically demonstrate that the algorithm runs faster when the weight vector has such characteristics.

The pseudocode of the algorithm is given below.

Algorithm 1 Top-k pruning method

```
Input: Database examples $\mathbf{x}_1, \ldots, \mathbf{x}_N$, weight vector $\boldsymbol{\theta}$, sorting indices $p_1, \ldots, p_D$ s.t. $|\theta(p_1)| \geq$
    $|\theta(p_2)| \geq \ldots > |\theta(p_D)|$.
Output: Indices of top-$k$ images: $A \subseteq \{1, \ldots, N\}$.
 1: Initialize candidate set: $C := \{1, \ldots, N\}$
 2: Set $A$ to contain the indices of $k$ randomly chosen images.
 3: $l^* := \sum_{d \text{ s.t. } \theta(d)<0} \theta(d)$
 4: $u^* := \sum_{d \text{ s.t. } \theta(d)>0} \theta(d)$
 5: $\forall i : u(i) := u^*, l(i) := l^*$
 6: for $d = 1$ to $D$ do
 7:     for all $i \in C$ such that $x_i(p_d) == 1$ do
 8:         if $\theta(p_d) \geq 0$ then
 9:             $l(i) := l(i) + \theta(p_d)$
10:         else {case $\theta(p_d) < 0$}
11:             $u(i) := u(i) + \theta(p_d)$
12:     for all $i \in C$ such that $x_i(p_d) == 0$ do
13:         if $\theta(p_d) \geq 0$ then
14:             $u(i) := u(i) - \theta(p_d)$
15:         else {case $\theta(p_d) < 0$}
16:             $l(i) := l(i) - \theta(p_d)$
17:     Update $A$ to contain indices of top-$k$ lower bounds
18:     Prune candidate set:
        $C = C - \{i \text{ s.t. } u(i) < min_{j \in A} l(j)\}$
19:     if $|C| == k$ then
20:         break
```

2.5 Efficient approximate ranking

The algorithm presented above achieves high efficiency by quickly removing from consideration images that cannot rank in the top-k. Instead, in this subsection we present an algorithm that performs fast retrieval by *approximating* the ranking score with a measure that can be computed efficiently. The exact score calculation is approximated via vector quantization. However, our descriptors are binary vectors, and as such they are not suited to be quantized. Thus, we first apply PCA to transform each binary-valued classeme vector $\mathbf{x}_i \in \{0,1\}^D$ into a real-valued lower-dimensional vector $\hat{\boldsymbol{x}}_i \in \mathbb{R}^{D'}$, where $D' < D$ [1]. Then, we quantize each vector $\hat{\boldsymbol{x}}_i$ using the product quantization method of Jégou et al. [32, 33]. This approach can provide very good vector approximation at low computational cost both during the learning of the cluster centroids as well as at quantization-time. The method splits each vector $\hat{\boldsymbol{x}}$ into v sub-vectors $\hat{\boldsymbol{x}}^1, \ldots, \hat{\boldsymbol{x}}^v$, each of length D'/v. Then, each sub-vector is quantized independently using a codebook of w cluster centroids learned from training data using k-means clustering. Thus, the complete vector $\hat{\boldsymbol{x}}$ is quantized as $q(\hat{\boldsymbol{x}})$ by the following quantizer function $q(.)$:

$$q(\hat{\boldsymbol{x}}) = \begin{bmatrix} q_1(\hat{\boldsymbol{x}}^1) \\ \vdots \\ q_v(\hat{\boldsymbol{x}}^v) \end{bmatrix} \tag{2.2}$$

where $q_j(\hat{\boldsymbol{x}}^j) \in \mathbb{R}^{D'/v}$ is the nearest cluster centroid to sub-vector $\hat{\boldsymbol{x}}^j$ in the dictionary learned for the j-th sub-block of features. While quantizers are usually employed to reduce the dimensionality of the data, we use them here primarily to speed-up the calculation of the score. Given the weight vector learned in the D'-dimensional space, the idea is to approximate the exact ranking score calculation $\hat{\boldsymbol{\theta}} \cdot \hat{\boldsymbol{x}}_i$ with $\hat{\boldsymbol{\theta}} \cdot q(\hat{\boldsymbol{x}}_i)$. Note that this approximate score can be computed as follows:

$$\hat{\boldsymbol{\theta}} \cdot q(\hat{\boldsymbol{x}}_i) = \sum_{j=1}^{v} \hat{\boldsymbol{\theta}}^j \cdot q_j(\hat{\boldsymbol{x}}_i^j) \,. \tag{2.3}$$

The efficiency stems from the fact that the terms $\hat{\boldsymbol{\theta}}^j \cdot q_j(\hat{\boldsymbol{x}}_i^j)$ can be read from a lookup table computed in a preprocessing stage for all v centroids of each sub-block j. The creation of this table for all sub-blocks will have cost of $\mathcal{O}(wD')$. But then computing the approximate score in eq. 2.3 will amount to simply adding together v values read from the look-up table. Thus, the overall complexity of calculating the ranking scores for all images in the databases, including the preprocessing, will be $\mathcal{O}(wD' + vN)$.

As discussed in full detail in [32], choosing the number of PCA dimensions D' poses a challenging dilemma. When D' is large, the PCA projection error is small, but there is a subsequent large quantization error. In principle this quantization error can be fought off by increasing v and w at the expense of a larger code size and a higher computational cost for quantization and learning. On the other hand, choosing a small D', leads to a large projection error followed by a small quantization error. In our problem the choice of D' has an even greater importance: since we are training our linear classifier in the PCA subspace, the choice of D' will dictate the Vapnik-Chervonenkis (VC) dimension, i.e., the capacity of our classification model [34]. A

[1] We also tried to use real-valued classeme vectors and achieved similar results. Here we prefer presenting the method based on binary classemes in order to compare the different methods in a scenario where they are all applied to the same input representation.

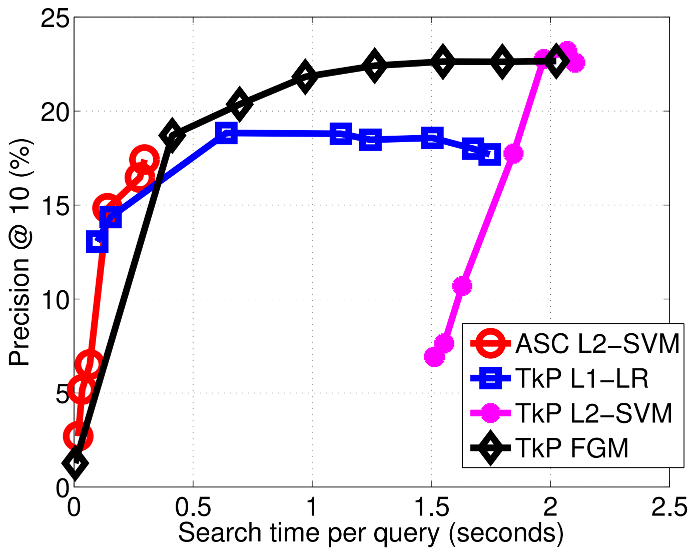


FIGURE 2.1: Class-retrieval precision versus search time for the ILSVRC2010 data set: x-axis is search time; y-axis shows percentage of true positives ranked in the top 10 using a database of 150,000 images (with $n^-_{test} = 149,850$ distractors and $n^+_{test} = 150$ true positives for each query class). The curve for each method is obtained by varying parameters controlling the accuracy-speed tradeoff (see details in the text).

linear classifier defined in a D'-dimensional space has VC dimension $D' + 1$. Thus, using a large D' will allow us to obtain more powerful classifiers. In the experiment section we analyze empirically how D', w, v affect the accuracy, the speed, as well as the memory usage.

Another practical issue to consider is that the PCA components, by construction, have different variance, with the first few entries typically capturing most of the energy in the signal. A naïve application of product quantization would subdivide a vector according to the order of components so that the j-th sub-block would consist of the consecutive feature entries from position $(1 + (j - 1)D'/v)$ to (jD'/v). However, such strategy would blindly allocate the same number of centroids for the most informative components (the ones in the first sub-block) as well as for the least informative. We address this problem using the solution proposed in [32]: we apply a random orthogonal transformation after PCA so that the variances of the resulting components will be more even. We then quantize the examples and train our retrieval models in this space.

2.6 Experiments

In this section we empirically evaluate the proposed algorithms and the several possible parameter options on challenging data sets under the performance measures of retrieval accuracy, speed and memory usage. We denote the top-k pruning method with **TkP** and the approximate ranking technique with **AR**.

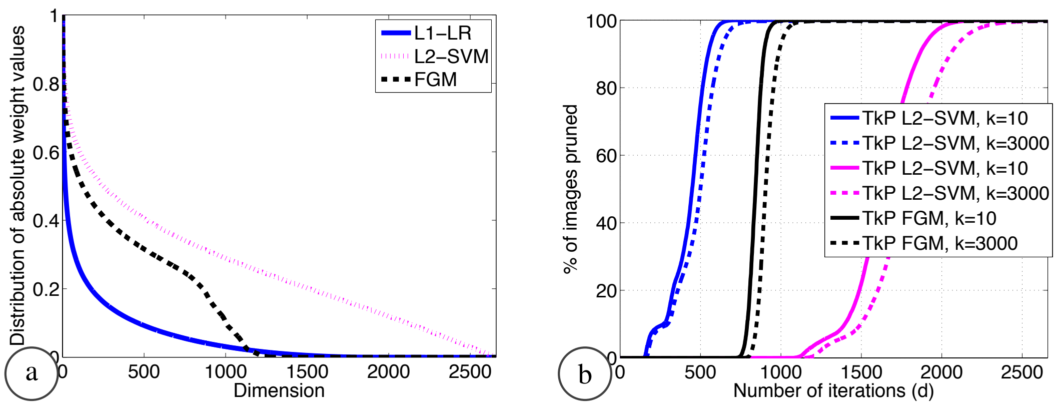


FIGURE 2.2: (a) Distribution of weight absolute values for different classifiers (after sorting the weight magnitudes). TkP runs faster with sparse, highly skewed weight values. (b) Pruning rate of TkP for various classification model and different values of k ($k = 10, 3000$).

Retrieval evaluation on ILSVRC2010 (150K images). We first evaluate our methods using the data set of the Large Scale Visual Recognition Challenge 2010 (ILSVRC2010) [35], which includes images of 1000 different categories. We use a subset of the ILSVRC2010 training set to learn the classifiers: for each of the 1000 classes, we train a classifier using $n^+ = 50$ positive examples (i.e., images belonging to the query category) and $n^- = 999$ negative examples obtained by sampling one image from each of the other classes. To cope with the largely unequal number of positive and negative examples ($n^- >> n^+$) we normalize the loss term for each example in eq. 2.1 by the size of its class. We evaluate the learned retrieval models on the ILSVRC2010 test set, which includes 150,000 images, with 150 examples per category. Thus, the database contains $n^+_{test} = 150$ true positives and $n^-_{test} = 149,850$ distractors for each query. Figure 2.1 shows precision versus search time for AR and TkP in combination with different classification models. Since AR does not use sparsity to achieve efficiency, we only paired it with the L2-SVM model. The x-axis shows average retrieval time per query, measured on a single-core computer with 16GB of RAM and an Intel Core i7-930 CPU @ 2.80GHz. The y-axis reports precision at 10 which measures the proportion of true positives in the top 10. The times reported for TkP were obtained using $k = 10$. The curve for AR was generated by varying the parameter choices for v and w, as discussed in further detail later. The performance curves for "TkP L1-LR" and "TkP L2-SVM" were produced by varying the regularization hyperparameter C in eq. 2.1. While C is traditionally viewed as controlling the bias-variance tradeoff, in our context it can be interpreted as a parameter balancing generalization accuracy versus sparsity, and thus retrieval speed. In the case of "TkP FGM" we have kept a constant C (tuned by cross-validation), and instead varied the sparsity of this classifier by acting on the separate parameter B. From this figure we see that AR is overall the fastest method at the expense of search accuracy: a peak precision of 22.6% is obtained by TkP using L2-SVM but AR with the same classification model achieves only a top precision of 17.5% due to a combination of fewer learning parameters (in this experiment we used $D' = 512$), PCA projection error and quantization error. As expected, we note that TkP runs faster when used in combination with L1-LR or FGM rather than L2-SVM, since it benefits from sparsity in the parameter vectors to eliminate images from consideration. However, we see that sparsity negatively affects accuracy, with L2-SVM providing clearly much better precision compared to L1-LR.

In our experiments we found that TkP typically exhibits faster retrieval in conjunction with L1-LR rather than FGM. We can gain an intuition on the reasons by inspecting the average distribution of weight absolute values in figure 2.2(a). The average distribution for each classification model was obtained by first sorting the weight absolute values for each query in descending order and then normalizing by the largest absolute value. For this experiment we chose $B = 1000$ for the FGM model. We can see that although for this setting the weight vectors learned by FGM are on average more sparse than those produced by L1-LR, the normalized magnitude of the L1-LR weights decays much faster. TkP benefits from the presence of these highly skewed weight magnitudes to produce more aggressive pruning. Figure 2.2(b) shows the average proportion of database pruned by the top-k method as a function of iteration number (d) for $k = 10$ and $k = 3000$. As anticipated, a smaller value of k allows the method to eliminate more images from consideration at a very early stage.

We now turn to study the effect of parameters D', v, w on the efficiency and accuracy of AR. Figure 2.3 shows retrieval speed and precision obtained by varying v and w for $D' \in \{128, 256, 512\}$. Increasing the dictionary size (w) reduces the quantization error while raising the quantization time: note the slightly better accuracy but higher search time when we move from parameter setting ($D' = 512, v = 256, w = 2^6$) to ($D' = 512, v = 256, w = 2^8$). The number of sub-blocks (v) critically affects the retrieval time: reducing v lowers a lot the search time but causes a drop in accuracy. Finally, note how D' impacts the accuracy since it affects both the number of parameters in the classifier as well as the projection error: using a large D' is beneficial for accuracy when v and w are large; however, when the number of cluster centroids is small, lowering D' improves precision since this mitigates the quantization error.

Finally, we also ran an experiment simulating real-world usage of an object-class retrieval system where a user may provide a positive training set but no negative set. In such cases one could use a "background" set for the negative examples. Thus, here we used as negative examples for each query, $n^- = 999$ randomly chosen images from all 1000 categories, thus possibly containing also some true positives (i.e., images of the query class). As expected, we found the precisions of the L1-LR and L2-SVM classifiers to be nearly unchanged by the few incorrectly labeled examples: precisions at 10 in this case are 18.75% and 22.55%, respectively.

Retrieval results on ImageNet (10M images). We now present results on the 10-million ImageNet dataset [36] which encompasses over 15,000 categories (in our experiment we used 15203 classes). We used a subset of 950 categories as query classes. For each of these classes

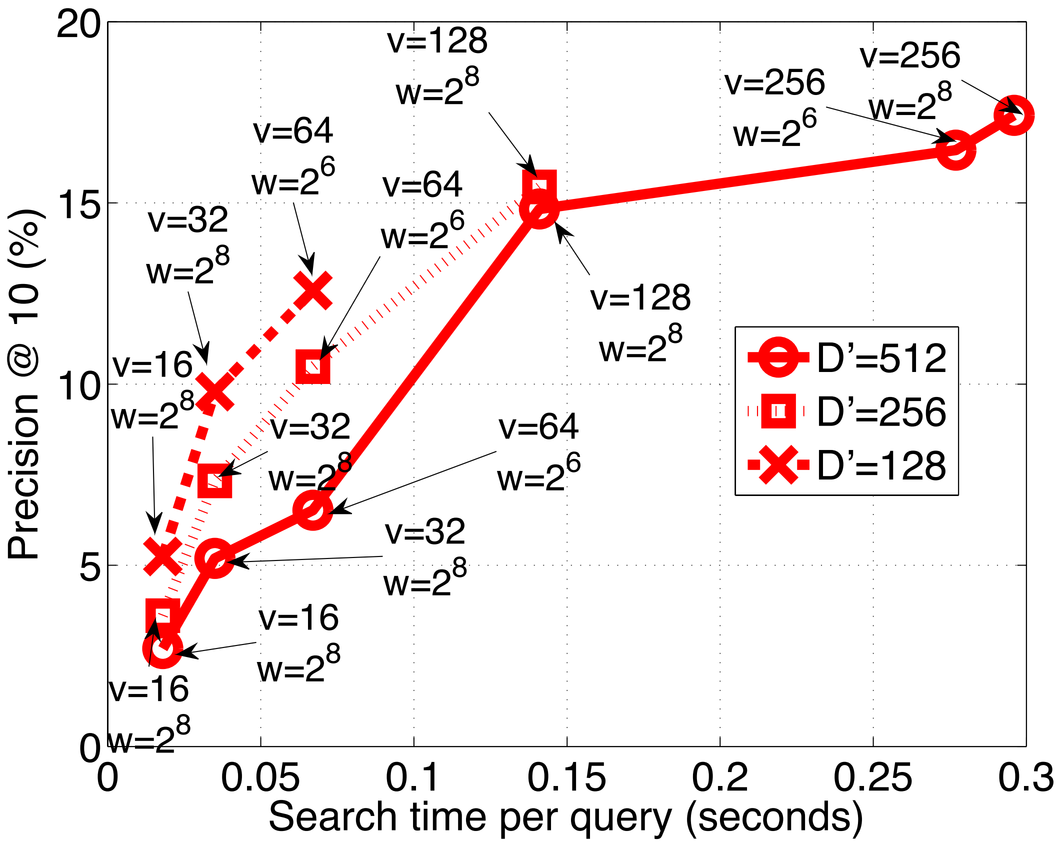


FIGURE 2.3: Effects of parameters D', v, w on the accuracy and search time of AR for the ILSVRC2010 data set. A small v implies faster retrieval at the expense of accuracy. Using a larger value for w reduces the quantization error at a small increase in search time. Lowering D' decreases the power of the classifier (VC-dimension) and increases the PCA projection error, thus negatively impacting precision.

we capped the number of true positives in the database to be $n+_{test} = 450$. The total number of distractors for each query is $n^{-}_{test} = 9,671,611$. We trained classifiers for each query category using a training set consisting of $n^{+} = 10$ positive examples and $n^{-} = 15,202$ negative images obtained by sampling one training image for each of the negative classes. We omit from this experiment the FGM model as its training time is over 300 times longer than the time needed to learn the L1-LR or L2-SVM classifier and thus its use on such a large scale benchmark is difficult (as a reference, learning a L1-LR or an L2-SVM classifier for a query category in this experiment takes around 2 seconds). The results are summarized in figure 2.4, once again in the form of retrieval time versus precision at 10. We can see that on this data set, TkP provides clearly the best accuracy-speed tradeoff with near peak-precision achieved for an average retrieval time of just a couple of seconds. The plot reports time for $k = 10$, but we found that when setting $k = 3000$ the retrieval time of TkP increases by only roughly 35% compared to the case $k = 10$. AR is once again very fast but it provided lower precision due to the issues pointed out above. In this figure we are also including the retrieval times obtained

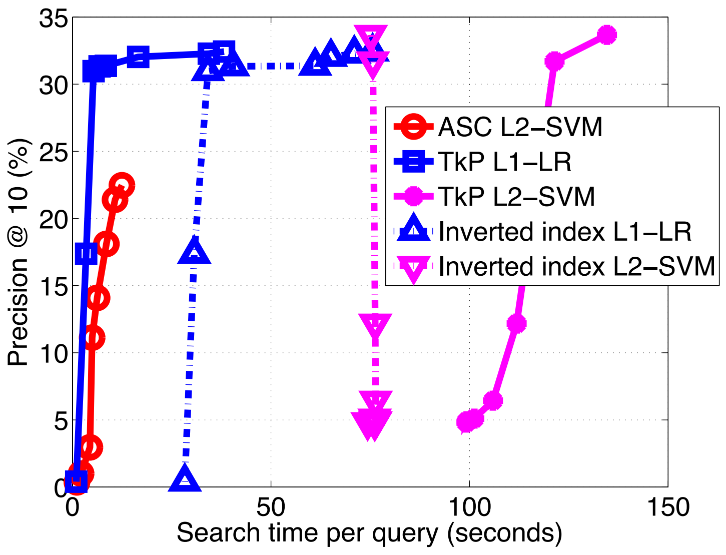


FIGURE 2.4: Search time versus retrieval precision at 10 for the 10-million ImageNet dataset. For each query class, there exist $n{+}_{test} = 450$ positive images and $n^{-}_{test} = 9,671,611$ distractors in the database.

with a simple architecture of inverted lists, with each list enumerating the images containing one particular classeme. Retrieval with inverted files obviously yields the same accuracy as TkP but it is more than 7 times slower. Overall, TkP with L1-LR provides a 24-fold speed-up compared to brute-force evaluation.

We would like also to comment on the memory usage. The inverted file architecture requires the most space. We represented the image IDs in inverted files using one byte per image: we achieve this by storing only ID displacements (which in our experiment happened to be always smaller than 255) between consecutive images in the list. Despite this clever encoding the total storage requirement for the 10M data set was roughly 9GB. TkP was implemented using a bit map of all classemes for all images which takes a space of $(2659/8) \times N$ bytes for a database containing N images, which in this case amounts to about 3GB. AR is the most space-efficient: it requires only $v \log_2 w$ bits to represent each image using vector quantization and the cluster centroids are stored in only $D'w$ real values. Thus on the 10M data set, the memory usage of AR was only 1.8GB. This is clearly the most scalable approach in term of memory usage.

Object-class retrieval accuracy on Caltech256. Our choice of retrieval models and features was primarily motivated by computational complexity constraints. Thus, a natural, legitimate question is: how much accuracy have we sacrificed for the sake of this efficiency? We answer this question by comparing the retrieval accuracy of our approaches with the state-of-the-art class-recognition system of Gehler and Nowozin [11], which has been shown to produce the best categorization results to date on several recognition benchmarks. This classifier combines non-linear kernel distances computed from multiple feature descriptors. Its high computational

	L1-LR	L2-SVM	FGM	LP-β
precision@25	28.2%	30.2%	29.6%	32.8%
training time (seconds)	0.044	0.028	11.04	253.7

TABLE 2.1: Caltech256 evaluation: precision at 25 and training time for the state-of-the-art LP-β classifier as well as for linear classifiers trained on binary classemes. The training set sizes are $n^+ = 50$ and $n^- = 255$. The number of true positives in the database is $n^+_{test} = 25$ and the number of distractors are $n^- = 6400$. The precisions of these simple linear models approach the accuracy of the LP-β classifier which is recognized as one of the best object classification systems to date.

complexity and large feature storage requirements makes it impossible to use in large image databases such a those considered in this paper. Thus, we carry out our this accuracy comparison on the small Caltech256 data set. We use as low-level features for LP$beta$ the same 13 descriptors that were used to learn the classemes [18], so as to have a comparison between methods on common ground. We train the retrieval models on each Caltech256 class separately by choosing $n^+ = 50$ positive examples of the query category and $n^- = 255$ negative examples obtained by sampling one image from each of the other categories. We report the precision on ranking a database of 6,400 images including $n^+_{test} = 25$ true positives and $n^-_{test} = 6,375$ distractors obtained by choosing 25 examples from each of the other 255 classes. Table 2.1 shows that our simple retrieval models applied to binary classeme vectors achieve accuracy comparable to that of the much more computationally expensive LP-β classifier and are several orders of magnitude more efficient to train as well as test.

Chapter 3

Learning Semantic Binary Codes (Attribute Discovery)

3.1 Overview

We describe a method that represents images with binary codes. In training, we infer codes for training images, and learn classifiers to predict the codes; in testing, we apply those classifiers to a test image to produce a code. An established literature shows how such binary codes can be used for image retrieval (e.g via hashing) and for image classification (e.g, via multi-class classification).

Image retrieval emphasizes appearance similarity, and many similar looking objects belong to the same category. This means that images that look similar (resp. dissimilar) should have similar (resp. dissimilar) codes. We call this property **unsupervised similarity**. These kinds of appearance similarity are not particularly discriminative. To ensure discrimination one needs to produce same (resp. dissimilar) codes for members of same(resp. different) categories. We call this property **discriminative similarity**. Our code construction balances unsupervised similarity with discriminative similarity.

Our codes are learned from category information on a per-image basis, meaning that training images within the same category may have different codes. We see this as an important innovation. It is natural, because not all objects within a category share all properties. Furthermore, doing this allows us to balance the discriminative information in a particular bit with our ability to predict the bit. One might try to train a system to predict a fixed code for each image within a category; however, there is no evidence that one can predict these codes accurately from images. Each bit in our codes can be thought of as a visual attribute whose name is not known. Like attributes, our codes lead to natural models of categories. Later in the process, we infer what is shared within each category by selecting the most informative bits per category.

Our experimental evaluations show that our codes can produce state of the art retrieval and classification performance results on Caltech256. We also perform extensive evaluations on ImageNet. Our bit codes behave like attributes, and we show that our method has discovered visual attributes (Figure 3.8). Like attributes, our bit codes can be used as features to recognize objects in categories not used for training the code (Figure 3.9). We show that learning codes on a per-image basis outperforms that of category-based codes. Finally, the space produced by our learned codes can model within category variations (Figure 3.10).

3.2 Background

Binary codes are attractive image representations for image search and retrieval, because they are easy to match, and the capacity of the space of very short binary codes is so large that all of the digital images in the world can be indexed with relatively short binary codes. 64-dimensional codes can index about 10^{19} images; 5 times the estimated number of bits created in 2002 and likely similar to the number of digital images in existence [37]. Unfortunately, it is not known how to perfectly encode visual information into the binary codes to enable efficient search and retrieval.

Binary codes have been usually used as hash keys where the hashing functions are learned to preserve some notion of similarity in the original feature space. Important examples include: locality sensitive hashing [38], where similar objects have high probability of collision; parameter sensitive hashing [39], where the hash code is adjusted to improve regression performance; kernelized locality sensitive hashing [40], which results in fast image search and retrieval; binary reconstructive embedding [41], which encourages distances between binary codes to mirror distances in the original image feature space; efficient retrieval [2]; and semantic hashing [42], which encourages distances between codes to mirror semantic similarities, approximated by category memberships. Semantic hashing methods can produce very efficient image search methods for collections of millions of images [43]. Semantic hashing methods use a restricted boltzman machine; extensions include stacking multiple restricted boltzmann machines [44]. Alternatively, Norouzi and Fleet [45] model the problem of supervised compact similarity-preserving binary code learning as a Latent SVM problem and defined a hashing-specific class of loss functions. None of these approaches would necessarily result in discriminative codes. In fact, Figure 3.5 shows that preserving patterns in the original feature space may hurt discrimination in both supervised and unsupervised methods. Our experiments demonstrate that our codes achieve significantly better performances compared to state of the art supervised and unsupervised binary code methods.

Rotating the original feature space, then quantizing, is another approach. In spectral hashing [46], compact binary codes are calculated by thresholding a subset of eigenvectors of the laplacian of the affinity graph. Spectral hashing has been shown to outperform RBM and boosting SSC in [39]. Raginsky and Lazebnik [47] project the data to a low dimensional space and then use random quantizations, after [48]. Lin et al [49] use an iterative learning method to produce binary codes whose hamming distance correlates to the similarities explained by the affinity matrix of the data in the original feature space; doing so requires long codes. Jégou et al. [50] jointly optimize for the search accuracy, search efficiency and memory requirements to obtain their binary codes. Gong and Lazebnik [51] iteratively minimize (ITQ) the quantization error of projecting examples from the original feature space to vertices of a binary hypercube. This method is capable of incorporating supervision by using CCA instead of PCA. ITQ has shown to produce state of the art results. Our experiments show that ITQ follows the patterns in the original feature space very well. This, however, may result in poor discrimination. We show that our binary codes consistently outperforms ITQ.

Many methods are unsupervised [46, 47, 51]. Some methods use a notion of similarity between labeled pairs of examples [42, 45, 49, 51]. Further, Wang et al. [52] get improvements over LSH and spectral hashing from a semi-supervised approach that minimizes the empirical loss over the labeled examples and maximizes the variance and independence of unlabeled examples. In either case, all methods assume that images that look "similar" should have the same label, but this is not always true for object categories.

Each bit in a binary code can be thought of as a **split** of the feature space into two half-spaces. Farhadi et al. [53, 54] construct thousands of random splits of the data, then pick the most predictable ones to generate random codes. Their splits are predictable and have high validation-set accuracy, but are not necessarily discriminative. Classeme features [55] are splits of data that produce state of the art results, but again there is no explicit discriminative component to the construction.

Alternatively, one could build codes out of object **attributes** [56] [57]. Such codes are easily interpreted semantically, but no explicit discriminative construction is yet known. Semantic attributes can also be discovered by selecting attributes that reduce the amount of confuision and are nameable [58, 59]. In terms of supervision, attributes are typically supervised or as recently shown they can be used in a semi-supervised fashion [60]. We do not use any supervision in terms of attributes. There is good evidence that random splits of data can produce informative bit strings [53, 55]. We differ from these constructions, because our method is explicitly discriminative. Furthermore, instead of learning bits independently, we learn bit vectors as a whole. Wang et al. [61] implicitly learns for discriminative codes by learning for hash functions that can sequentially correct the mistakes of the previous codes. We differ from them, because we

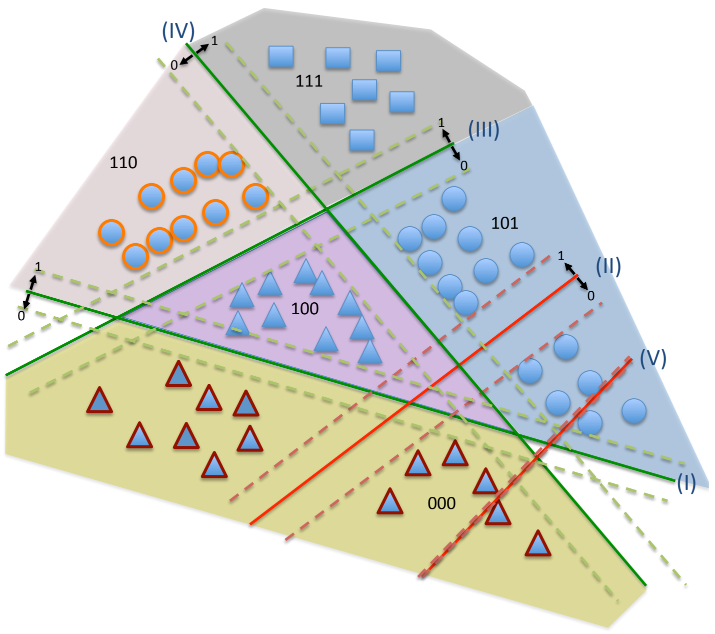


FIGURE 3.1: Each bit in the code can be thought of as a hyperplane in the feature space. We learn arrangements of hyperplanes in a way that the resulting bit codes are discriminative and also all hyperplanes can be reliably predicted (enough margin). For example, the red hyperplanes (II,V) are not desirable because II is not informative(discriminative) and IV is not predictable (no margin). Our method allows the green hypeplanes (good ones) to sacrifice discrimination for predictability and vice versa. For example, our method allows the green hyperplane (I) to go through the triangle class because it has strong evidence that some of the triangles are very similar to circles.

explicitly optimize for discrimination in a max margin framework, we learn for the binary codes as a whole (not one by one), and we optimize jointly for discrimination and predictability.

3.3 Learning Discriminative Binary Codes

Our goal is to learn codes for each instance in the training set such that a) the codes can be reliably predicted from the visual data and b) if we represent each image with its learned codes then discrimination becomes easier. Our system consists of two main parts: learning binary codes for each instance and then performing search or classification in the space of binary codes. For learning our binary codes, we optimize for two criteria jointly: we want our codes to be as discriminative as they can, while maintaining predictability of the codes. The most discriminative codes (like assigning unique codes to examples of the same category) are extremely hard to predict from visual data. And the most predictable codes may contain very little information about categories resulting in poor discrimination. Our model balances between discrimination and predictability of the codes. In our view a code is discriminative if examples of different categories appear far away from each other and instances of the same category lie close by. However, we don't enforce these discriminative constraints as hard constraints but assign codes to each image in a way that the resulting codes have enough discriminative power and yet can be reliably predicted from images. Such a code allows for simple, efficient and very accurate classification and searches. For performing search and classification in the space of binary codes we use KNN and linear SVM. In Section 3, we demonstrate that KNN search in the space of our binary codes outperforms KNN on other state-of-the-art binary code spaces on Caltech256. We also show that linear SVM classifiers using our codes results in even higher accuracies with very few training examples per category.

Throughout this work we use the term "splits" when we refer to bits of a code. Each bit can be visualized as a hyperplane that separates instances that have value 0 versus the ones that have value 1. Each bit of our codes is generated by checking which side of a hyperplane an instance lie. In [53], the splits are learned by randomly setting a subset of examples to positive and another subset to negative. Some of these splits can be reliably predicted from data. In [53], the splits are sorted based on how well they can be predicted from the data. The top K splits produce a K dimensional binary code. This procedure does not necessarily result in discriminative binary codes and the codes may need to be very long to ensure good performance. We believe the procedure to learn the splits and the procedure to find good binary codes should be learned jointly. This results in binary codes that are predictable and have built-in margins. This means that each code is associated with a cell in an arrangement of hyperplanes in the feature space (Figure 3.1). The family of such arrangements is rich, meaning that we can find good codes.

Assume that we are given a training set $\{x_i, y_i\}$ where $i \in \{1, 2, ..., N\}$, $x_i \in \ R^d$, and $y_i \in \{1, 2, ..., C\}$ and we plan to learn k splits, meaning that our final binary code is k-dimensional. To train each split $s \in \{1, 2, ..., k\}$ we have to learn for labels to give us positive and negative training examples l^s where $l^s \in \{-1, 1\}$, -1 for negative and 1 for positive training examples. For each instance i in the training set, we are learning for l_i^s indicating which side of the split s the i^{th} example should appear. When learning for these codes our goal is to learn for a set of labels for each instance in a way that those labels can be reliably predicted from visual data and the learned codes leave enough space between categories. The final binary codes are the actual predictions of each split classifier s trained with l^s as training labels. We call these predictions $b_i^s \in \{0, 1\}$. b_i^s indicates which side of the split s the i^{th} example actually lies. In other words, b_i^s is the prediction of a classifier that uses l_i^s as training labels; l is what we want and b is what we can actually predict. We can stack all the labels and the predictions from all the split classifiers to form the final binary vector for each instance i in the training set: $L_i = [l_i^1, l_i^2, ..., l_i^k]$, and $B_i = [b_i^1, b_i^2, ..., b_i^k]$.

We are looking for binary codes that (a) can be reliably predicted from the original features and (b) provide enough margins between examples from different categories. To do that we learn to allocate codes to instances by searching for the whole code that jointly optimizes these two criteria. We do not optimize bits one by one as we can not guarantee uncorrelated binary codes. Our formulation looks like a combination of max margin models for linear SVMs to satisfy (a) and pushing for large inter class and small intra class distances for (b). We achieve such codes by optimizing:

$$\begin{aligned}
&\min_{w,\xi,L,B} \frac{1}{2} \sum_{c\in\{1:C\}} \sum_{m,n\in c} d(B_m, B_n) + \gamma \sum_{s\in\{1:k\}} \|w^s\|^2 \qquad (3.1)\\
&+\lambda_1 \cdot \sum_{\substack{i\in\{1:N\}\\ s\in\{1:k\}}} \xi_i^s - \frac{\lambda_2}{2} \sum_{\substack{c'\in\{1:C\}\\ p\in c'}} \sum_{\substack{c''\in\{1:C\}\\ c'\neq c'', q\in c''}} d(B_p, B_q)\\
&s.t. \quad l_i^s(w^{s'} x_i) \geq 1 - \xi_i^s \quad \forall i \in \{1:N\}, s \in \{1:k\}\\
&\qquad b_i^s = (1 + sign(w^{s'} x_i))/2 \quad \forall i \in \{1:N\}, s \in \{1:k\}\\
&\qquad \xi_i^s > 0 \quad \forall i \in \{1:N\}, s \in \{1:k\}
\end{aligned}$$

where d can be any distance in the hamming space, $B_i = [b_i^1, b_i^2, ..., b_i^k]$, w^s is the weight vector corresponding to the s^{th} split, ξ_i^s is the slack variable corresponding to the s^{th} split and i^{th} example, C is the total number of categories, k is the number of splits, N is the total number of examples in the train set, l_i^s is the training label for the i^{th} example to train the s^{th} split, and b_i^s indicates the prediction results of i^{th} example using the split s.

This is an extremely hard optimization problem, but we may not need to find the global minimum to obtain good binary codes. "Good" local minima are capable of producing promising discriminative binary codes. To go down the objective function in the optimization 1 we use an iterative block coordinate descent method. In algorithm 6 we described our optimization steb-by-step.

We initialize by choosing B to form orthogonal codes that come from projections along PCA directions. In our experiments we find that this initialization yields promising results. The supplementary material describes the (minor) effects of other choices of initialization. We then initialize w^s to predict these codes. Notice that the w's are independent given a fixed B, so we can use an SVM.

We now proceed by iterating three steps in sequence. **First**, we update B for fixed w_i^s, ξ_i^s; this proposes an improvement in the codes that should achieve improved separation. This is an iterative procedure that is started at the current value of B. We use stochastic gradient descent (step 4) with an important optimization. Since B is binary, if b_i^s is 0 then the sum of differences is the number of 1s and vise versa. We can pre-compute number of 0s and 1s for each s^{th} element of B. This way, we decrease the complexity of computing sum of differences from $O(N^2K)$ to

Algorithm 2 Optimization

```
Input: $X = [x_1, ...x_N]$ ($x_i$ is low-level feature vector for $i^{th}$ image).
Output: $B$ ($B_i = [b_i^1, b_i^2, ..., b_i^k]$ is binary code for $i^{th}$ image ).
1: Initialize $B$ by: $B \leftarrow$ Projection of $X$ on first $k$ components of $PCA(X)$
2: Binarize $B$: $B \leftarrow (1 + sign(B))/2$
3: repeat
4:     Optimizing for $B$ in $\min_B \frac{1}{2} \sum_{c \in \{1:C\}} \sum_{m,n \in c} d(B_m, B_n) - \frac{\lambda_2}{2} \sum_{c' \in \{1:C\} p \in c'} \sum_{c'' \in \{1:C\} c' \neq c'', q \in c''} d(B_p, B_q)$
       (see supplementary materials for details )
5:     $l_i^s \leftarrow (2b_i^s - 1) \; \forall i \; i \in \{1 : N\}, \forall s \; s \in \{1 : k\}$
6:     Train $k$ linear-SVMs to update $w^s \; \forall s, \; s \in \{1 : k\}$ using $L$ as training labels ($l_i^s$ : label for $i^{th}$ image when
       training $s^{th}$ split)
7:     $b_i^s \leftarrow (1 + sign({w^s}^T x_i))/2 \;\; \forall i, s \; i \in \{1 : N\}, \; s \in \{1 : k\}$
8: until Convergence on optimization 1
```

$O(NK)$. **Second**, we update L using B and then (Fixing L, B) we update w_i^s, ξ_i^s by training SVMs using L as training labels. This produces a set of SVM's to predict these improved codes. Each bit of B represents a labeling of instances that we want an SVM to reproduce. We can therefore compute optimal w_i^s and ξ_i^s with an SVM code. **Third**, we update the current value of B to reflect the codes that these SVM's actually predict; this biases the update of B in the direction of codes that can be predicted. While this optimization procedure doesn't guarantee descent in each iteration, we have found that we get descent in practice (Figure 3.3). This is most likely because the steps balance the goodness of the code (updated in the first step), with our ability to predict it (second, third steps). In our experiments, we did not tune the parameters; λ_1,γ are set to 1 and λ_2 is set to normalize for the size of categories. Figure 3.3 shows the behavior of the objective function and all the terms in equation 1 after each iteration.

Once converged, optimization 1 provides us the weight vectors w^{s*} for split classifiers that tend to produce binary codes with built-in margins. We use w^{s*} to project the data to the space of the binary codes.

Using Codes: There are several ways to use the resulting binary codes. We evaluate our codes in a) using them as hash codes and performing KNN on the codes (called KNN in our experiments), b) using them as features and learning SVM classifiers for each category (called SVM), c) using the codes as features while accepting that these features might be redundant and using L1 regularized models to pick category specific codes (e.i. for each category we learn a L1-regularized SVM and pick the bits correspond to larger absolut weight value of the L1-SVM) and then learn normal SVM classifers using related bits.

3.4 Experimental Evaluations

Tasks: The main tasks of our experiments are in classification and category retrieval. We compare our method in several different settings with the state of the art bit-code-based methods. We also compare our method with state of the art classification techniques. Our bit learning

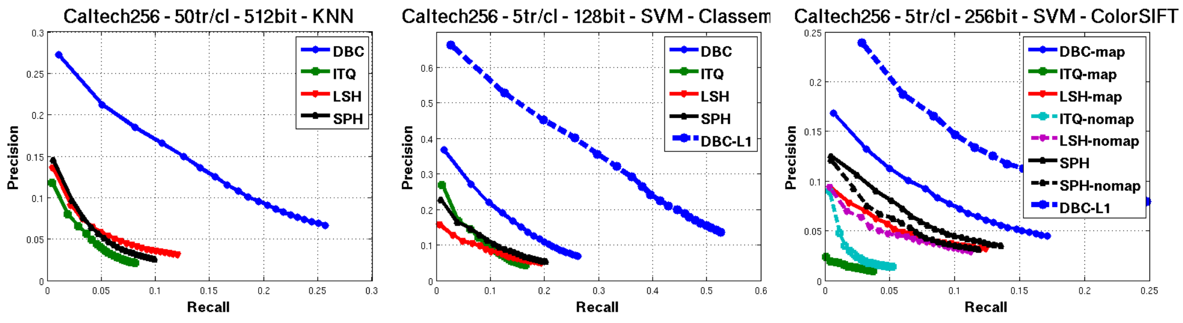


FIGURE 3.2: We compare our binary codes(DBC) with Locality Sensitive Hashing(LSH), Spectral Hashing(SpH), and supervised version of Iterative Quantization(ITQ) under several different settings: changing the length of binary codes (32,64,128,256), classifiers (linear SVM or KNN), original features (Classeme, ColorSift) and also with L1 selection of category specific bits (DBC-L1). Our codes (DBC) consistently outperforms state of the art methods like SpH and ITQ by large margins. The test set contains 25 examples per category. Due to space limitations only very few of experimental settings can be showed. Please see supplementary material for all plots.

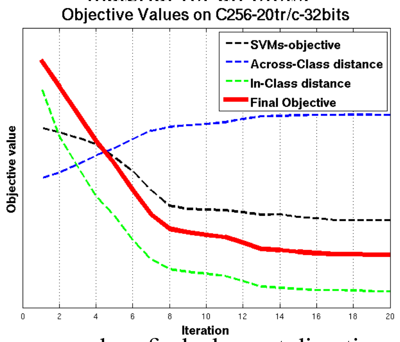


FIGURE 3.3: Our optimization procedure finds descent directions in our challenging objective function. This figure shows that all terms in the objective function actually improves after each iteration.

algorithm results in interesting observations about the data like attribute discovery. Also, we qualitatively evaluate our method in retrieval and attribute discovery. Our method is also applicable to novel category recognition.

Datasets: We test our method on Caltech256 [62], and ImageNet [63] (ILSVRC2010). Both of these dataset have large number of categories (256 and 1000) with huge intra-class variations. Category retrieval on Caltech256 is a challenging task because the number of categories is much higher than typical experiments and also the intra-class variations are much higher than typical datasets like MNIST. There are around 30000 images belonging to 256 categories [62]. On average, there are about 120 images per category.

Features: For experiments on Caltech256 we use two different sets of features that have been shown to produce state-of-the-art results on Caltech256: Classeme and ColorSift. We use Classeme features [55] because they have shown to outperform other features [55]. The Classeme features are of 2659 dimensionality. We also use ColorSift features as they show promising performances on classification tasks [64]. We use ColorSift bag-of-words features by building a

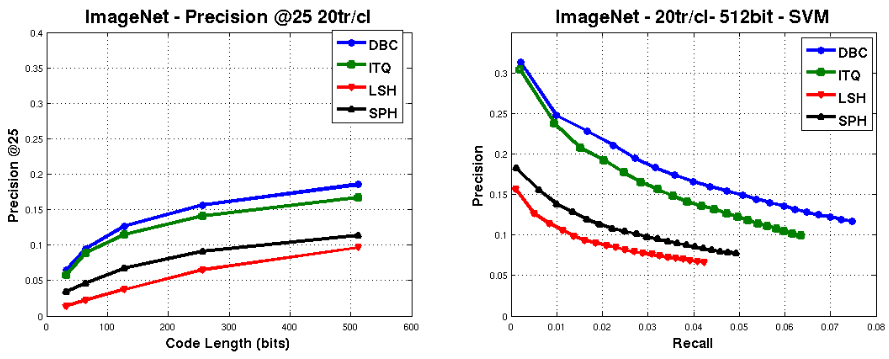


FIGURE 3.4: Our method outperforms state of the art binary code methods (LSH, SpH, and supervised TTQ) on ImageNet(1000 categories, 20 per catgeory for training). Left plot compares precesion @25 vs. the code length. The test set contains 150 images per category. The right plot shows precision-recall curve for the same dataset using 512 dimensional codes. Our codes consistently outperforms all other methods.

1000-word dictionary using ColorSift features provided by [65]. To make these features more discriminative we use homogeneous kernel map [66] on top of SIFT-BoW. The homogeneous kernels have shown to produce best results in many classification tasks [66]. Both of these features are among the most discriminative features. For ImageNet experiments we also use Classeme features.

Controls: To evaluate our method we perform series of extensive evaluations and comparisons. For our method, we change the following settings: the length of binary codes k (32, 64, 128, 256, 512), the number of training examples per category (5 0r 50), the original features (Classemes or ColorSift), the classifier (LSVM [67] or KNN), and the use of L1 selection of category specific bit strings. To compare with methods in the literature we compare our results with Locality Sensitive Hashing(LSH) as a standard baseline, with the supervised version of Iterative Quantization (ITQ) [51] as the best supervised method and Spectral Hashing (SpH)[46] as the state-of-the-art unsupervised method in producing binary codes. Our experimental evaluations demonstrate that our method consistently outperforms state-of-the-art methods under all the combinations of above settings. To evaluate our method on a large scale dataset we test it on ImageNet. We used 1000 category from ILSVRC2010 (ImageNet Challenge). For each category we randomly chose 20 examples for training and 150 examples for testing. Our results show that our codes also outperform state-of-the-art binary code results on this dataset.

Measurements: In case of SVM, we use the top k images to compute precision and recall values. Varying $k = [1 : 5 : 100]$ traces out the precision-recall curves. In case of using KNN, for each number of nearest neighbors we can compute a precision and recall. Varying this number makes a precision recall curve.

Results: There are four main categories of results. First, we compare our method with the state of the art bit-code methods on Caltech256 and ImageNet. We also show interesting qualitative results. Second, we compare our results with the state of the art method on Caltech256. Third, we compare our method on novel category recognition with the state of the art method of [68].

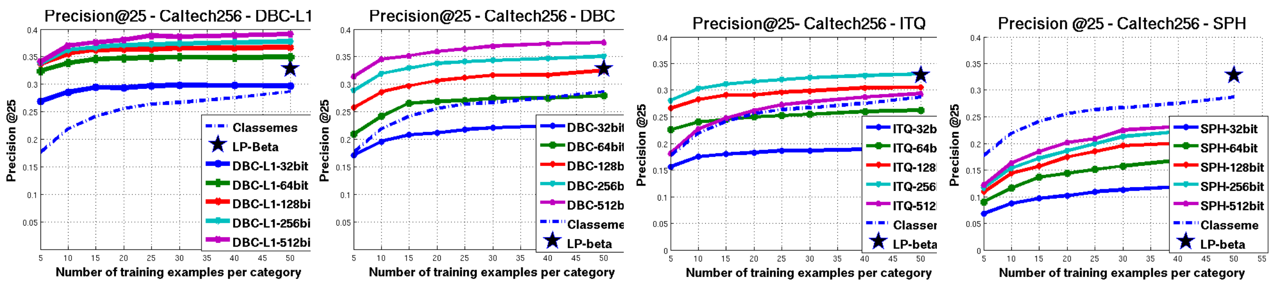


FIGURE 3.5: Our method produces state of the art results on Caltech256. A linear SVM with only 128-bit code is as accurate as multiple kernel learning method of LPBeta (marked with a big star) that uses 13000 dimensional features. As we increase the size of the code we outperform the LPBeta method significantly. This figure compares our category specific codes (DBC-L1) , our codes without L1 selection (DBC), ITQ and SPH on precesion at 25 versus the number of training examples per category on caltech 256. One interesting observation is that the ITQ method does a great job in following the original features (Classeme) with 512 codes. This however hurts the performance as 128 and 256 dimensional codes outperforms the original features. This confirms our intuition that following the patterns in the original feature space does not necessarily result in good performance numbers.

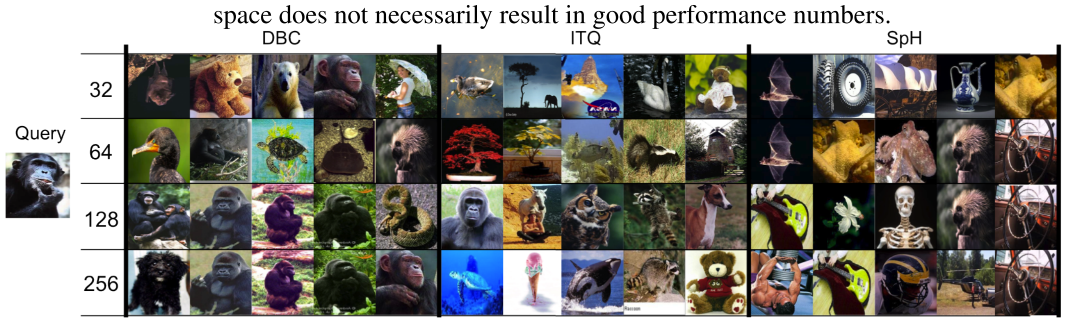


FIGURE 3.6: This figure qualitatively compares the quality of retrieved images by our method comparing to that of ITQ and SpH. Each row corresponds to the top five images returned by three different methods: ours, ITQ and spectral hashing. This retrieval is done by first projecting the query image to the space of binary codes and then running KNN in that space. Notice how, even with relatively short codes(32 bits), our method recovers relevant objects. This menas that the discriminative training of the code has forced our code learning to focus on distinctive shared properties of categories. Our method consistently becomre more accurate as we increase the code size.

Fourth, we show qualitative results that reveal interesting properties of our method. We show promising attribute discovery results and also projections of the resulting bit code space.

Comparisons to the state of the art bit-code methods: Figure 3.2 compares our method (DBC, DBC-L1) with LSH,SpH, and supervised version of ITQ by varying the number of binary codes. We perform extensive evaluations on all combinations of different settings. Space does not allow showing all comparisons in all settings, please see supplementary material for all comparisons. The settings that we show here are: (from left to right on Figure 3.2) using KNN on 512-dimensional bit coses when 50 training examples per category are observed during training using Classeme features, using SVM on 128-dimensional bit codes when 5 training examples per category is observed during training using Classeme features, and using SVM on 256-dimensional bit codes when 5 training examples per category is observed during training

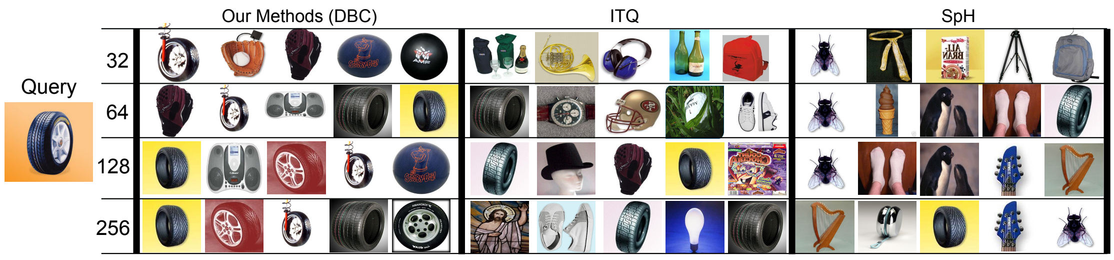


FIGURE 3.7: This figure qualitatively compares the quality of retrieved images by our method comparing to that of ITQ and SpH. Each row corresponds to the top five images returned by three different methods: ours, ITQ and spectral hashing. This retrieval is done by first projecting the query image to the space of binary codes and then running KNN in that space. Notice how, even with relatively short codes(32 bits), our method recovers round objects. Our method is consistent in terms of returned images as we increase the code size. With 256 dimensional code our method returns 5 correct images.

using ColotSift features. In all possible settings, including these three, our method outperforms state of the art bit code methods. We also show that DBC-L1 performs better than DBC in all settings. The gap between the DBC and DBC-L1 increase as the number of bits decreases. The huge gap in the lower number of bits is due to the fact that in DBC-L1 we chose the bits to be specific at each category. In all of the experiments we use the same random selection of train and test set.

Our experiments show that as we increase the neighborhood size in KNN our method can still find the right categories (see supplementary materials). This implies that our hash cells remain pure as we increase the size of the neighborhood. This confirms that the optimization 1 managed to produce codes with enough margins. It is also worth noting that with such small training set per category linear SVM achieves excellent results using our codes.

In figure 3.5 we compare all the methods in terms of the precision at top-25 ranked images with different code length. We also compared our method with product quantization [69] for 5tr/cl and follow the same experimental setup. Product quantization got the precision of 0.04, 0.05, 0.064, 0.08, 0.09 for 32, 64, 128, 256, 512 bits repectively. Our method outperforms all the methods in all different code lengths. The Left plot in Figure 4 shows this comparison on ImageNet. For all other comparisons on Caltech256 please see the supplementary material. In these experiments, the test set contains 25 images per category . Figure 3.6 and 3.7 qualitatively compare our discriminative binary codes with ITQ and SpH in an image retrieval task. We show the top five retrieved images for the query image. It is interesting to see that even with 32 dimensional code our method is capable of extracting relevant properties. Our method is consistent in terms of returned images as one increases the code size.

Comparison to the state of the art models on Caltech256: Figure 3.5 compares our results with state-of-the art methods on Caltech256. We use the same features as the state-of-the-art

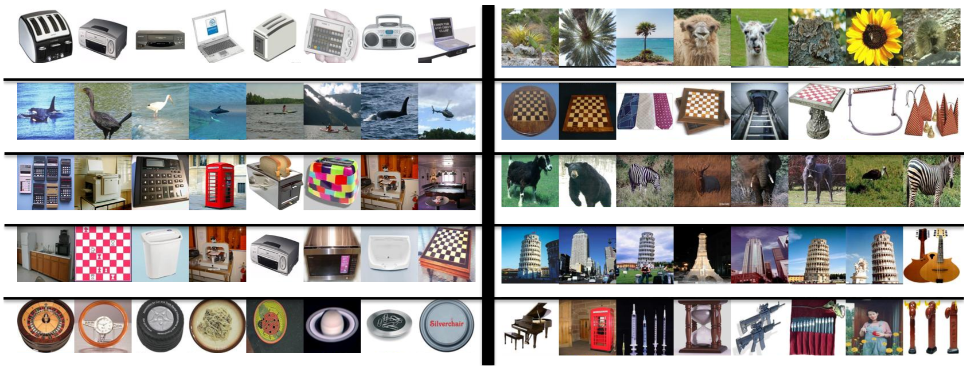


FIGURE 3.8: Discovering attributes: Each bit corresponds to a hyperplane that group the data according to unknown notions of similarity. It is interesting to show what our bits have discovered. On two sides of the black bar we show 8 most confident images for 5 different hyperplanes/bits (Each row). Note that one can easily provide names for these attributes. For example, the bottom row corresponds to all round objects versus objects with straight vertical lines. The top row has silver, metalic and boxy objects on one side and natural images on the other side, the second row has water animals versus objects with checkerboard patterns. Discovered attributes are in the form of contrast: both sides have its own meaning. These attributes are compact representations of standard attributes that only explain one property. For more examples of discovered attributes please see supplementary material.

method of LPBeta (The big star in the figure). With only 128 bits we can achieve the same results as the state of the art method of LPBeta that uses 13000 dimensional features. By increasing the number of bits our codes outperform the multiple kernel learning method of LPBeta. This shows that DBC can be significantly more discriminative than state-of-the-art features. We also compare with the classeme features. In this Figure we perform the same test with other binary code method. ITQ is doing a great job in getting close to the original features of Classeme by using 512 binary codes. However, it gets worse comparing to using 128 or 256 codes. This is mainly due to the fact that ITQ minimizes the quantization error of binarization and this does not necessarily result in better discrimination. Our method consistently gets better with more and more bits.

Novel Category Recognition: So far, we have shown that our codes are discriminative for categories they have been trained on. Similar to cross category generalization of attributes, we also evaluate our method on categories that have not been observed during training. For that, we learn the binary codes on 1000 categories of ImageNet with 20 examples per category and test our codes on Caltech256. We make sure that none of the 1000 categories intersect with 256 categories on Caltech. We adopt an experimental setting from [68] for which training data is available. Figure 3.9 shows that our method outperforms PiCodes [68], the state of the art novel category recognition method. We used the same low-level features as in [68].

Attribute Discovery: Binary codes can be though of as attributes. Our algorithm discovers attributes that can be named without much difficulty. Figure 3.8 shows some of the attributes discovered by our method. Each row shows 8 most confident examples for both sides of a hyperplane that corresponds to a bit in our code. Our learning procedure can discover attributes

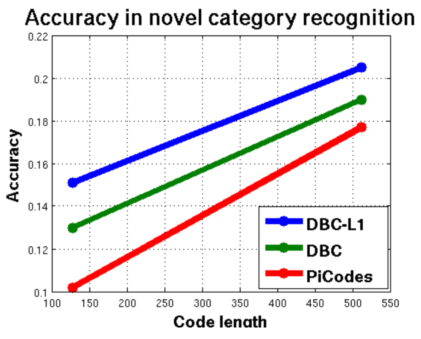


FIGURE 3.9: Our codes can be used across the trainining categories (novel categories): we use 1000 categories of ImageNet to train our codes and use the codes to recognize objects in Caltech 256. The 1000 categories from ImageNet do not intersect with those of Caltech256. Our method outperforms state of the art methods in novel categories.

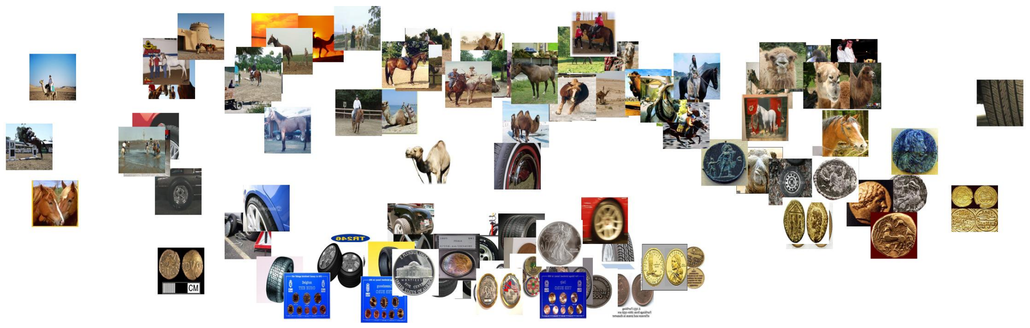


FIGURE 3.10: Projection of the space of binary codes: We use multidimensional scaling and project our 64 dimensional codes into a two dimensional space. It is interesting to show that our method clearly balances between discrimination and learnability of the codes: round objects like wheel and coins appear close by while horses and camels are faraway. The head of the horse and the head of camels are close to each other and far way from side views of them. Supplementary material includes more examples of these projections.

like is round, is boxy, is natural, has checkerboard pattern, and etc. More discovered attributes can be find in supplementary material. Our model learns strong contrasts that are discriminative. As a result of this each side of the discovered attributes has its own meaning. The discovered attributes are compact versions of standard attributes. Standard attributes describe only one property. But our discovered attributes are in the form of contrasts. For example, the first row contrasts boxy and silver objects against natural objects. If the bit that corresponds to the first row is 1 this means that the attribute boxy is 1 and if the bit is zero this means that the attribute natural is 1. It is also very interesting to look at the space of binary codes. To do that, we project our binary codes into a 2-dimensional space using multidimensional scaling. Figure 3.10 shows an interesting balance between discrimination and classification. In the projected space round things like wheels and coins are close together despite belonging to different categories. At the same time, round things are far away from horse and camel examples. Examples of the head of horse and camels are closer together than those to side views of horses and camels. Category memberships are suitable proxies for visual similarity but should not be enforced as hard constraints. Our model manages to balance between discrimination in terms of basic level categories and learnability of the codes from visual data.

Chapter 4

Predictable Dual-View Hashing

4.1 Overview

Binary codes are attractive representations of data for search and retrieval purposes due to their efficiency in computation and storage capacity. For example, 64-bits binary codes can index about 10^{19} images, five times the estimated amount of data created in 2002 and quite likely the total number of digital images in existence [37].

Hashing is a common method for assigning binary codes to data points (*e.g.*,, images). The binary codes are used as hash keys where the hash functions are learned to preserve some notion of similarity in the original feature space. Such binary codes should have the general hash property of low collision rates. In addition, suitable binary codes for search and retrieval should also maintain high collision rates for similar data points. The latter property is essential in a similarity based retrieval settings [51, 70, 71].

The binary codes can be learned either in a unsupervised manner that models the distribution of samples in the feature space [71] or in a supervised manner that uses labels of the data points [72]. Unsupervised methods can be adversely affected by outliers in distributions and noise, and the supervised methods require expensive manual labeling.

It is often the case that information about data are available from two or more views, *e.g.*,, images and their textual descriptions. It is highly desirable to embed information from both domains in the binary codes, to increase search and retrieval capabilities. Utilization of such binary codes will create a cross-view Hamming space with the ability to compare information from previously incomparable domains. For example in the text and image domain, image-to-image, text-to-image, and image-to-text comparisons can be preformed in the same cross-view space. Such approaches have received attention recently due to the emergence of large amounts of data in different domains being available on the internet.

To date, most approaches proposed embedding dual-views in Hamming space use canonical correlation analysis (CCA) [73–75]. The CCA based approaches are less sensitive to feature noise and require no manual labeling. However, bits learned by CCA do not explicitly encode the proximity of samples in the original feature space since CCA enforces orthogonal bases and aims to reduce the modality gap with little consideration of the underlying data distribution.

To address this issue, we propose a dual-view mapping algorithm that represents the distribution of the samples with non-orthogonal bases inspired by a notion of *predictability* proposed in [3]. Predictable codes ensure that small variations of the data point positions in the original space should not result in different binary codes. In other words, a particular bit in the binary code should be identical (predictable) for all data samples that are close to each other in each view. To maintain such predictability, we employ a max-margin formulation that enforces confident prediction of bits.

Furthermore, we propose a joint formulation for learning binary codes of data from two different views. We assume that a latent Hamming space exists for the data, and optimize the hash functions that map the data from each view to this common space, while maintaining the predictability of the binary codes. Knowing the hash functions in the original views supports cross-modal searches.

The rest is organized as follows: Section 5.2 reviews related work. Section 5.3 presents the details of our approach including optimization methods. Experimental analysis and comparisons to state-of-the-art methods are presented in section 10.4.

4.2 Background

As our work lies in the intersection of hashing methods and mutil-view embedding, we briefly describe related work in both domains. We also review specific applications that could be enabled via our method.

Gionis *et al.* [38] introduced Locality Sensitive Hashing (LSH) where similar objects have high probability of collision. Along this direction, Shaknarovich *et al.* [39] use parameter sensitive hashing and apply it to human pose estimation. Kulis and Grauman [40] extend LSH with kernels and show fast image search for example-based searches and content based retrieval. Kulis and Darrell [41] also proposed a binary reconstructive embedding method for minimizing the differences between Euclidean distances in the original feature space and the Hamming distances in the resulting binary space.

Semantic hashing, proposed in [42], learns compact binary codes that preserve correlations between distances in the Hamming space and semantic similarities approximated by category memberships. This is accomplished by learning a deep generative model, called a Restricted Boltzman Machine (RBM) which has a small number of nodes in a deepest level that produce a small number of binary values. Torralba *et al.*[43] extend this idea to efficient image search method on the scale of millions of images. Nonlinear mapping to binary codes has been addressed in [44] by stacking multiple RBM's. Norouzi and Fleet [45] model the problem of supervised learning of compact similarity-preserving binary code using a Latent SVM problem and define a hashing-specific class of loss functions. None of these approaches, however, necessarily captures the semantics of an image. In fact, enforcing preservation of patterns in the original feature space may hurt discrimination in both supervised and unsupervised methods.

Utilization of textual captions for image understanding has recently received considerable attentions in the research community. Farhadi *et al.* [76] introduce a CRF based method to model a semantic space that text and images can be mapped to via triples of object, subject and verb. In [77] strategies of creating image-text datasets via Amazon Mechanical Turk are investigated. Kulkarni *et al.* [78] propose a method for generating natural language descriptions from images by parsing a large set of texts and performing object recognition on image sets. Li *et al.* [79] propose a simple but effective N-gram based method that can produce simple descriptions of pictures. The generated descriptions are not identical to the text corpora, *i.e.*,, they compose a sentence entirely from scratch. Recently, several works presented methods for Multi-Modal hashing [80–82]; most of them having high computational complexity which limits their applicability.

Ordonez *et al.* [83] created a large-scale dataset of images and captions, and proposed a method for generating textual captions for images from this dataset. A method for recognition of visual texts and non-visual texts is proposed in [84]. Kuznetsova *et al.* [85] use multiple noisy captions for images from the web and combine them to produce a more meaningful sentence for an image. Berg *et al.* [86] approach the problem of text generation to emphasize the visually salient aspects of an image.

4.3 Dual-View Hashing

Without loss of generality, we assume that the two views are visual (image) and textual (description). However, our approach is applicable to any domain, and this assumption only facilitates the discussion.

We use the following notation; $X_{\mathcal{V}}$ represents data in the visual space and $X_{\mathcal{T}}$ indicates data in the textual space. X_* is a $d_* \times n$ matrix whose columns are vectors corresponding to the points in either spaces. d_* is the dimension of either visual or textual space which might be different. x_*^i is the i^{th} column of X_*. $*$ is a placeholder for $\mathcal{V}$ or $\mathcal{T}$.

4.3.1 Dual-View Embedding

Our goal is to find two sets of hyperplanes $W_{\mathcal{V}}, W_{\mathcal{T}} \in \mathbb{R}^{d_* \times k}$ (k is the dimension of the common subspace, *i.e.,*, length of binary code) that map the visual and textual space into a common subspace. Each hyperplane (each column of W_*) divides the corresponding space into two subspaces; each point in a space is represented as **-1** or **1** depending on which side of the hyperplane it lies in. w_*^i indicates the i^{th} column of W_*. Among the infinite possible hyperplanes, the ones that binarize the points in the visual space and the textual space consistently are desirable for our purpose. This objective can be achieved by minimizing the following function:

$$\min_{W_{\mathcal{V}}, W_{\mathcal{T}}} \|\text{sign}(W_{\mathcal{V}}^T X_{\mathcal{V}}) - \text{sign}(W_{\mathcal{T}}^T X_{\mathcal{T}})\|_2^2 \tag{4.1}$$

However, Eq.(4.1) is a non-convex combinatorial optimization problem; it has a trivial solution when both $W_{\mathcal{V}}$ and $W_{\mathcal{T}}$ are zero. To avoid the trivial solution and force each bit to carry the maximum amount of information, we add constraints to enforce low correlation of the bits. With these constraints, we can reformulate the problem as:

$$\begin{aligned} \min_{W_{\mathcal{V}}, W_{\mathcal{T}}} & \|W_{\mathcal{V}}^T X_{\mathcal{V}} - B_{\mathcal{T}}\|_2^2 + \|B_{\mathcal{T}} B_{\mathcal{T}}^T - I\|_2^2 \\ & + \|W_{\mathcal{T}}^T X_{\mathcal{T}} - B_{\mathcal{V}}\|_2^2 + \|B_{\mathcal{V}} B_{\mathcal{V}}^T - I\|_2^2 \\ s.t. & \\ & B_{\mathcal{T}} = \text{sign}(W_{\mathcal{T}}^T X_{\mathcal{T}}) \\ & B_{\mathcal{V}} = \text{sign}(W_{\mathcal{V}}^T X_{\mathcal{V}}) \end{aligned} \tag{4.2}$$

where minimizing $\|B_* B_*^T - I\|_2^2$ enforces low correlation of bits. This optimization cannot be directly solvable, but it can be solved approximately by relaxing B_* [87] and applying CCA [73], which leads to the following generalized eigenvalue problem:

$$\begin{pmatrix} S_{\mathcal{V}\mathcal{V}} & S_{\mathcal{V}\mathcal{T}} \\ S_{\mathcal{T}\mathcal{V}} & S_{\mathcal{T}\mathcal{T}} \end{pmatrix} \begin{pmatrix} w_{\mathcal{V}} \\ w_{\mathcal{T}} \end{pmatrix} = \lambda \begin{pmatrix} S_{\mathcal{V}\mathcal{V}} & 0 \\ 0 & S_{\mathcal{T}\mathcal{T}} \end{pmatrix} \begin{pmatrix} w_{\mathcal{V}}, \\ w_{\mathcal{T}}, \end{pmatrix} \tag{4.3}$$

where $S_{\mathcal{V}\mathcal{T}} (= X_{\mathcal{V}} X_{\mathcal{T}}^T)$ is the covariance matrix between visual and textual features and w_* is a column of W_*.

Although CCA can find the underlying subspace, binarizing data in this subspace by $\text{sign}(W_*^T X_*)$ suffers from high quantization error. To reduce the quantization error, an iterative method is proposed in [51] that searches for a rotation of data points. Their approach, however, is not applicable to more than one domain. In addition, the approach assumes orthogonality of all of the projected hyperplanes, *i.e.,*, the columns of W_*. But the orthogonality is not always necessary

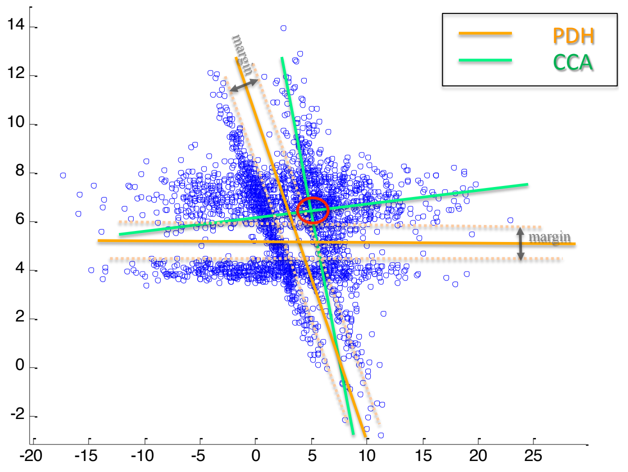


FIGURE 4.1: Comparison of learned hyperplanes by our method (PDH) and canonical correlation analysis (CCA). Note that the hyperplanes learned by the PDH divide the space, avoiding the fragmentation of sample distributions by the help of *predictability* constraints implemented by max-margin regularization.

and sometimes harmful. In contrast, we replace orthogonality of the hyperplanes by the notion of predictability of binary codes in the following section.

4.3.2 Predictability

Predictability is the ability to predict the value of a certain bit of a sample by looking at that bit of the nearest neighbors of that sample. For example, if the i^{th} bit in most of the nearest neighbors of a sample is **1** then we would predict that the i^{th} bit of that sample would be also **1**.

Consider the situation where a hyperplane crosses a dense area of samples; there would be many samples in proximity to each other that are assigned different binary values in the bit position corresponding to that hyperplane. Such binary values obtained by that hyperplane are not *predictable*. Intuitively, the binary values determined by a hyperplane are *predictable* when the hyperplane has large margins from samples. Figure 7.2 illustrates the hyperplanes determined by CCA in green lines in a 2D single domain (view). Note that CCA hyperplanes cross dense areas of samples and are orthogonal to each other whereas our PDH hyperplanes do not. If we binarize the samples by CCA hyperplanes, samples in the red circle will have different binary codes from each other, even though they are strongly clustered. The hyperplanes that are shown by orange lines represent our method (PDH), which enforces large margins from samples.

To learn the predictable W, we regularize the formulation with max-margin constraints. In fact, we learn multiple SVMs in visual space with respect to training labels in the textual space and vice versa. The final objective function is:

$$
\begin{aligned}
&\min_{W_{\mathcal{V}}, W_{\mathcal{T}}, \xi_{\mathcal{V}}, \xi_{\mathcal{T}}} \|B_{\mathcal{T}} B_{\mathcal{T}}^T - I\|_2^2 + \|B_{\mathcal{V}} B_{\mathcal{V}}^T - I\|_2^2 + \\
&\qquad \sum \|w_{\mathcal{V}i}\| + \sum \|w_{\mathcal{T}i}\| + C_1 \sum \xi_{\mathcal{V}} + C_2 \sum \xi_{\mathcal{T}} \\
&s.t. \\
&\quad B_{\mathcal{T}} = \text{sign}(W_{\mathcal{T}}^T X_{\mathcal{T}}), \\
&\quad B_{\mathcal{V}} = \text{sign}(W_{\mathcal{V}}^T X_{\mathcal{V}}), \\
&\quad B_{\mathcal{T}}^{ij} ({w_{\mathcal{V}i}}^T X_{\mathcal{V}}^j) \geq 1 - \xi_{\mathcal{V}}^{ij} \quad \forall i, j, \\
&\quad B_{\mathcal{V}}^{ij} ({w_{\mathcal{T}i}}^T X_{\mathcal{T}}^j) \geq 1 - \xi_{\mathcal{T}}^{ij} \quad \forall i, j.
\end{aligned}
\tag{4.4}
$$

Despite the complex appearance of the optimization, it is a perfect setting for block-coordinate descent and can be solved by an Expectation Maximization (EM) iterative algorithm. A detailed description of our iterative algorithm is as follows:

First, we fix all the variables except $W_{\mathcal{V}}$ and $\xi_{\mathcal{V}}$. Then we solve for these variables, which is multiple linear SVMs; one for each bit. To learn the i^{th} SVM, we use columns of $X_{\mathcal{V}}$ as training data and the elements of the i^{th} row of $B_{\mathcal{T}}$ as training labels. **Second**, using the outputs of these SVMs, $W_{\mathcal{V}}$, we compute $B_{\mathcal{V}} = \text{sign}({W_{\mathcal{V}}}^T X_{\mathcal{V}})$. **Third**, we update $B_{\mathcal{V}}$ to minimize the correlation between bits via minimizing $\|B_{\mathcal{V}} B_{\mathcal{V}}^T - I\|_2^2$. Since this problem is not trivial to solve, we use spectral relaxation [71] by creating a Gram matrix $S = B_{\mathcal{V}}^T B_{\mathcal{V}}$ and a $n \times n$ diagonal matrix $D(i,i) = \sum_j S(i,j)$ as the relaxed problem:

$$
\begin{aligned}
&\min_{B_{\mathcal{V}}} \text{tr}(B_{\mathcal{V}}(D - S){B_{\mathcal{V}}}^T) \\
&s.t. \quad B_{\mathcal{V}} {B_{\mathcal{V}}}^T = I.
\end{aligned}
\tag{4.5}
$$

The solutions are the k eigenvectors of $D - S$ with minimal eigenvalues, which we binarize by taking the sign of the elements. **Fourth**, we run the same three steps to compute $W_{\mathcal{T}}$. We repeat all the steps until convergence of the objective function. More details of the algorithm are provided in Algorithm 6

For initializing values for optimization, we tried several random values and the values obtained using CCA. But the results are not sensitive to the initialization, since in each block coordinate descent step, the objective function is convex. Thus, we use the values obtained by CCA for all initializations.

Since our objective function is not convex and we use block coordinate descent to optimize, the solution we obtain is not the global minimum. But our experiments suggest that the obtained local minima is good enough.

Algorithm 3 Predictable Dual-View Hashing

```
Input: $X_\mathcal{V}, X_\mathcal{T} \in \mathbb{R}^{d_* \times n}$.
Output: $B_\mathcal{V}, B_\mathcal{T} \in \mathbb{B}^{d_* \times k}$.
 1: $W_\mathcal{V}, W_\mathcal{T} \in \mathbb{R}^{d_* \times k} \leftarrow CCA(X_\mathcal{V}, X_\mathcal{T}, k)$
 2: $B_\mathcal{V} \leftarrow \text{sign}({W_\mathcal{V}}^T X_\mathcal{V})$
 3: $B_\mathcal{T} \leftarrow \text{sign}({W_\mathcal{T}}^T X_\mathcal{T})$
 4: repeat
 5:     $W_\mathcal{V} \leftarrow$ Weights of $k$ linear SVMs (for $i^{th}$ SVM: training features are columns of$X_\mathcal{V}$ and training labels are
        elements of $i^{th}$ row of $B_\mathcal{T}$)
 6:     $B_\mathcal{V} \leftarrow \text{sign}({W_\mathcal{V}}^T X_\mathcal{V})$
 7:     Update $B_\mathcal{V}$ using Eq. (4.5)
 8:     $W_\mathcal{T} \leftarrow$ Weights of $k$ linear SVMs (for $i^{th}$ SVM: training features are columns of$X_\mathcal{T}$ and training labels are
        elements of $i^{th}$ row of $B_\mathcal{V}$)
 9:     $B_\mathcal{T} \leftarrow \text{sign}({W_\mathcal{T}}^T X_\mathcal{T})$
10:     Update $B_\mathcal{T}$ using Eq. (4.5)
11: until convergence
12: $B_\mathcal{V} \leftarrow \text{sign}({W_\mathcal{V}}^T X_\mathcal{V})$
13: $B_\mathcal{T} \leftarrow \text{sign}({W_\mathcal{T}}^T X_\mathcal{T})$
```

4.4 Experiments

First, we show that our optimization algorithm solves the proposed objective functions. Then for the empirical validation, we present both quantitative and qualitative results for image category retrieval. In the quantitative analysis, we perform image classification and compare the mean average precision (mAP) obtained by our method with several state-of-the-art binary code methods. In qualitative analysis, we show that the sets of images retrieved by our binary code with both image and text queries contain semantically similar images. Our MATLAB software is available[1].

4.4.1 Datasets and Experimental Setup

For the dual-view situation, we need a dataset of images that are annotated with sentences. We use two datasets; PASCAL-Sentence 2008 introduced by [76] (one view is visual and the other is textual) and a recently collected large scale dataset, SUN-Attribute database (one view is visual and the other is semantic (attribute)) [88].

4.4.1.1 PASCAL-Sentence Dataset 2008

The images in the PASCAL-Sentence dataset are collected from PASCAL 2008, which is one of the most popular benchmark datasets for object recognition and detection. For each of the 20 categories of the PASCAL 2008 challenge, 50 images are randomly selected; in total, there are 1,000 images in the dataset. Each image is annotated with 5 sentences using Amazon's Mechanical Turk. These sentences represent the semantics of the image.

[1]http://umiacs.umd.edu/ mrastega/pdh/

Image Features: Our image features, following [76], are collections of responses from a variety of detectors, image classifiers and scene classifiers. Given an image, we run several object detectors on the image and set the threshold low enough so that each fires at least in one location. Then, we report the location of the most confident detector along with the confidence value. If we have 20 detectors, for each of the detectors we report $[x_i, y_i, c_i]$ which x_i,y_i are the coordinate of the location at which the detectors fired and c_i is the confidence value for that detector. Image and scene classifiers are SVMs trained on each category of objects on the global low-level GIST descriptor [89].

Text Feature: Text features are also from [76]. We construct a dictionary of 1,200 words from the sentences of the entire dataset that are frequent and discriminative with respect to categories. There are no prepositions and stop words in the dictionary. Let us call this set S. For a given sentence, we go through each word and compute its semantic similarity with all the words in S as a feature for that word. As a feature of the sentence, we simply sum all the vectors in each sentence. The semantic distance between two words is computed by the Lin similarity measure [90] on the WordNet hierarchy.

4.4.1.2 SUN Attribute Dataset

The SUN-Attribute dataset is a large-scale dataset [88] that includes 102 attribute labels annotated by 3 Amazon Mechanical Turk worker for each of the 14,340 images from 717 categories, which is a subset of the scene images from the SUN Dataset [91]. In total, there are four million (4M) labels. For each of 717 categories, there are 20 annotated scenes.

Image Features: We use the precomputed image features used in [88, 91], *i.e.,*, Gist, 2×2 Histogram of Oriented Gradient, self-similarity measure, and geometric context color histograms.

Attribute Features: Each image has 102 attributes and each attribute has multiple annotations. In total, there are four million labels that are annotated by Amazon Mechanical Turk workers with bad-worker filtering and good-worker cultivating strategies [88]. Some examples of annotated attributes are vegetation, open area, camping, hiking, natural light, leaves *etc.*

4.4.1.3 Experimental details

We use Liblinear [92] to learn SVMs for learning W_*. The parameters used for linear SVMs are $C_1 = 1$ and $C_2 = 1$ in Eq. 4.4. We did not tune those parameter. We also used linear SVM for category retrieval. We reduce the dimensionality of visual features in the SUN dataset from 19,080 to 1,000 by PCA.

4.4.2 Optimization Analysis

As we use a block coordinate descent algorithm to optimize the objective function, we cannot guarantee that our algorithm reaches the global optimum. Our experiments shows that we reach a reasonable local optimum most of the time. To illustrate this, we measure the objective value and see if it decreases (*in the minimization task*) or not. In figure 4.2, we observe that the objective values does decrease as the iterations go on. After only a few iterations (15) the differences between the textual binary codes (*binary codes extracted from text data*) and the visual binary codes (*binary code extracted from images*) are very small- less than 3 bits. The number of bits we use for this experiments is 32.

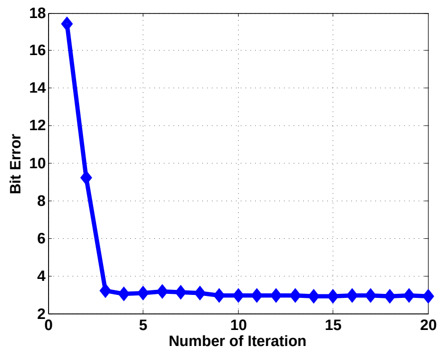


FIGURE 4.2: The objective function of Eq.(4.1) decreases as iterations of our block coordinate descent continue. 'Bit Error' refers to the number of bits that differ in the obtained binary codes from two different views. (32bit code learning)

4.4.3 Bit Error by Hamming Space Size

We investigate the Hamming distance of two obtained binary codes (value of Eq. 4.1) as a function of binary code length; 16, 32, 64, 128 and 256. Figure 4.3 shows that the number of bits that differ between binary codes from visual and text domains is almost always approximately $\frac{1}{10}$ of code length.

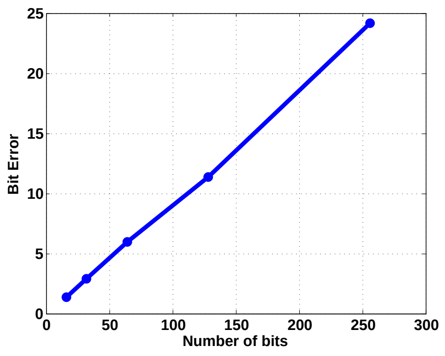


FIGURE 4.3: The error between textual and visual binary codes is a linear function of the length of the binary code. 'Bit Error' refers to the number of bits that differ in the obtained binary codes from two different views.

4.4.4 Image Category Retrieval

We retrieve images from an image pool by giving one or more samples (image or text/attribute) of a particular category as a query. In quantitative analysis, we compute the mean average precision (mAP) of retrieved images that belong to the same category of the query. In qualitative analysis, we present the images retrieved for a query by our method.

4.4.4.1 Quantitative Results

For quantitative analysis, we conduct a category retrieval experiment similar to [2, 3, 55]. We divided the dataset into two train/test segments. We train W_* using the training set. We compute the binary features for all the images (train and test). We take a set of images of a particular category as query set and train a classifier by taking the query set as positive samples and images from other categories in the training set as negative set. Then, we apply the classifier to all the samples of the test set, rank them by their classification confidence value and retrieve the top-K samples. We report precision and recall as an accuracy measure. By varying K in top-K we can draw a precision-recall curve. Since we are considering multiple categories, we report mean precision and recall.

We compare our binary code with several binary code methods including Iterative Quantization (**ITQ**) [51], Spectral Hashing (**SH**) [71] and Locality Sensitive Hashing (**LSH**) [38]. Our method is referred to as Predictable Dual-view Hashing (**PDH**). We are not comparing our method with [3] because their method is not applicable to Dual-View. They require category labels of samples as supervision to train their binary codes. We used supervised ITQ coupled with CCA which uses data in two views to construct basis vectors in a common subspace.

Figure 4.4 and Figure 4.5 show mean average precision (mAP) of retrieved images by our method and other methods as a function of the number of bits. We presents the results with various numbers of queries given. As shown in the figure, our method (**PDH**) consistently outperforms all other methods.The high ranked images are not necessarily visually similar to the query. When we have few instances in the retrieval set the baseline methods have better precision because the high ranked images are the most visually similar to a query. This is not unexpected, since we optimize for cross-domain similarity, not visual similarity. We can directly compare by average precision(AP). As recall increases and the number of relevant images from the database tat are visually similar to the query are exhausted, the PDH dominates the other methods in precision.

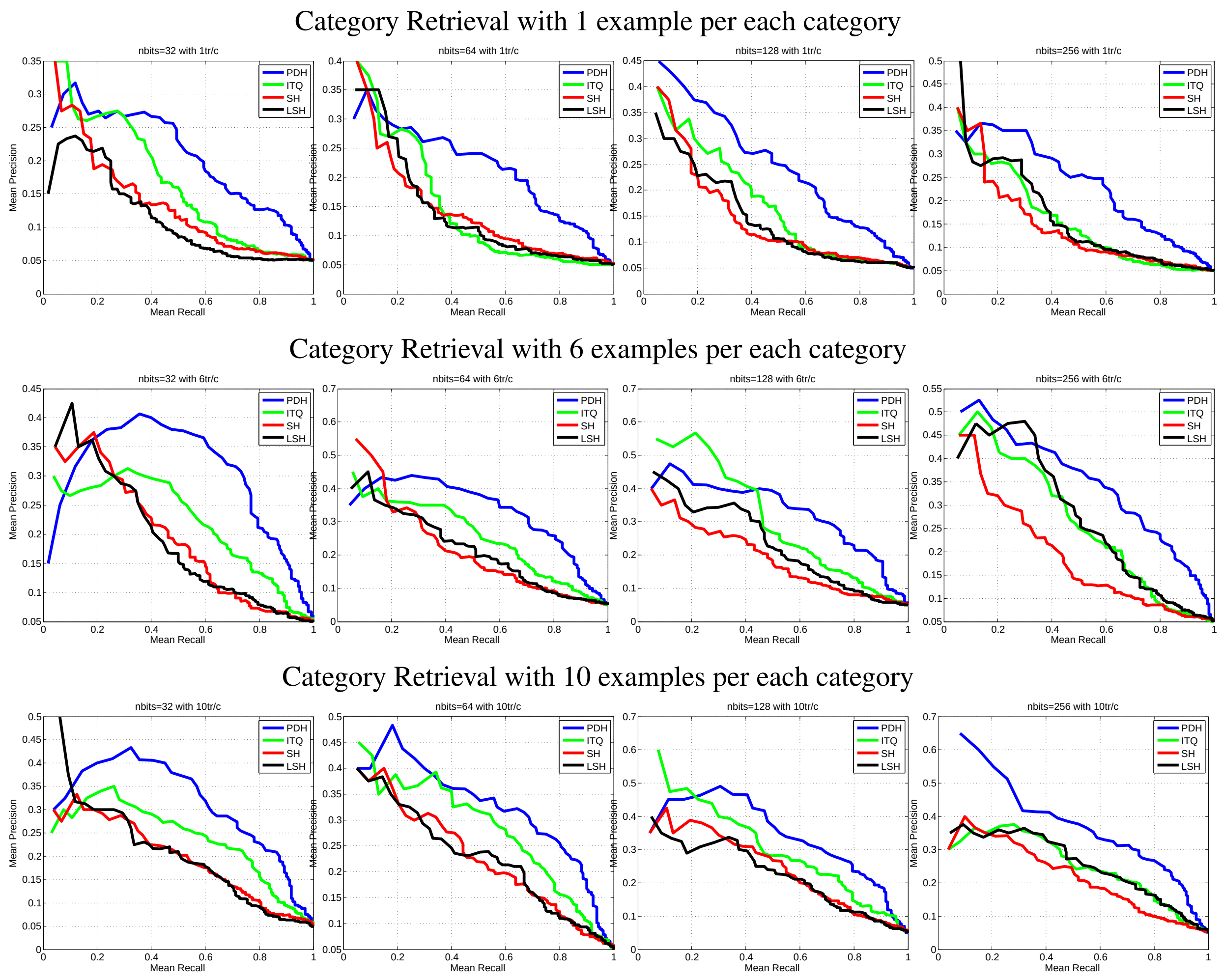


FIGURE 4.4: The result of category retrieval on PASCAL-Sentence dataset. Our method (PDH) is compared with three other baselines , Iterative Quantization (ITQ), Spectral Hashing (SH) and Locality Sensitive Hashing (LSH). We run experiments under different settings. We vary the code length (32, 64, 128 and 256) and we also vary the number of examples per each category in query by (1, 6 and 10)

4.4.4.2 Qualitative Results

We also present qualitative results of how our binary code performs. We perform two qualitative evaluations.

First, we conduct **Image2Image** retrieval. Given an image as a query, we retrieve the top-K closest images. Unlike the previous experiment we do not use an SVM but simply compute the Hamming distance of all other samples to the query sample and report the top-k most similar. Figure 4.6-(a) shows the retrieval for four query images which are represented by 32 bits. We report the top-5 most similar images. These retrieved images have significant semantic similarity to their query image.

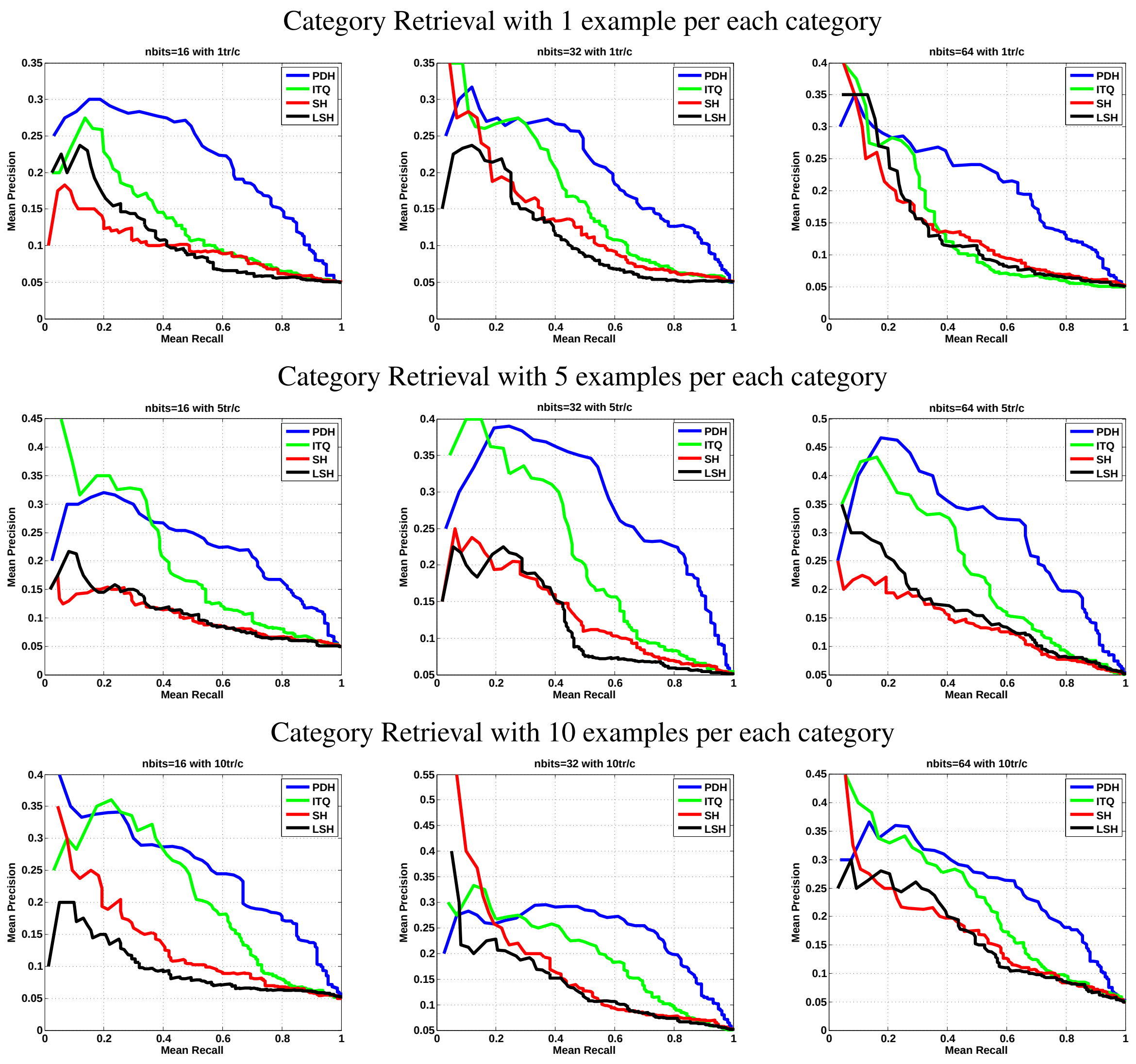
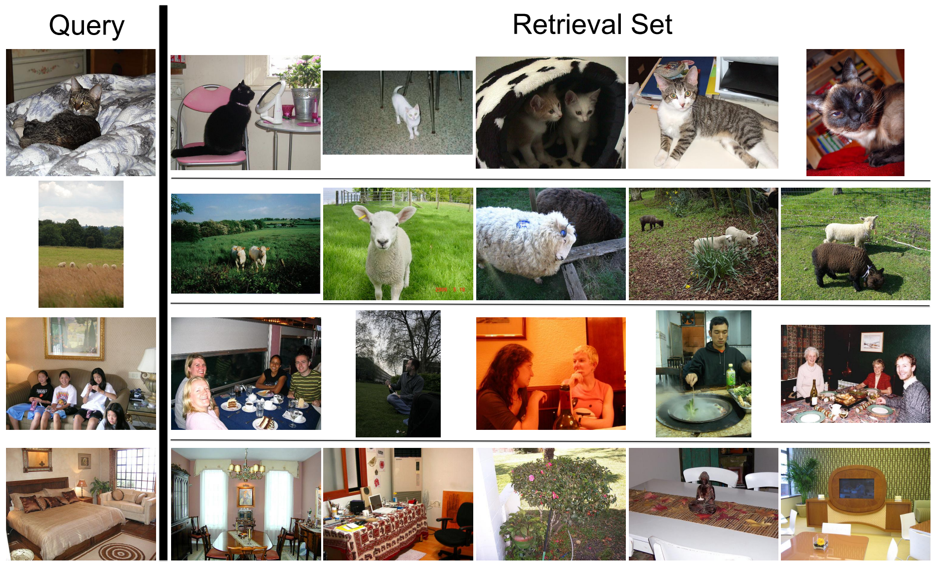
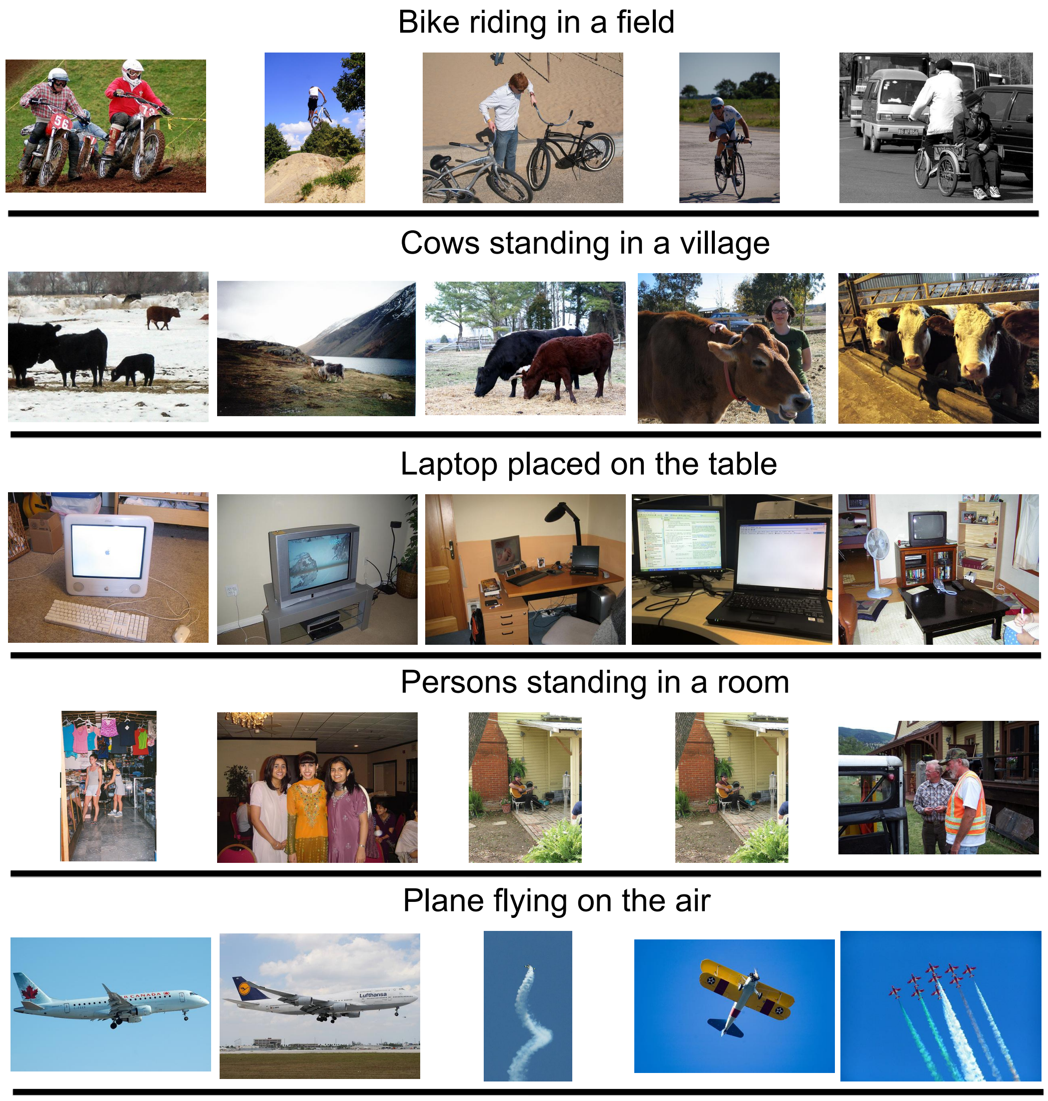


FIGURE 4.5: The result of category retrieval on SUN Dataset. Our method (PDH) is compared with three other baselines , Iterative Quantization (ITQ), Spectral Hashing (SH) and Locality Sensitive Hashing (LSH). We run the experiment under different settings of the problem. We changed the code length (32, 64, 128 and 256) and we also changed the number of examples per each category in query by (1, 6 and 10)

Second, we perform a **Text2Image** retrieval task. Instead of using an image as query we use a sentence as query and we retrieve images for which this query sentence could be a good description. We map the sentence to our binary space and then identify similar points (images) in that space and report the top-k most similar. In figure 4.6-(b) we illustrate the retrieval set for five different sentences using 32 bit codes. Most of the retrieved images have content that is semantically similar to their query sentence.




Bike riding in a field
Cows standing in a village
Laptop placed on the table
Persons standing in a room
Plane flying on the air

(a)

(b)

FIGURE 4.6: (a) **Image2Image** retrieval. Given an image as a query, we find most similar images by nearest neighbor search in 32 bit PDH. (b) **Text2Image** retrieval. Given a sentence as query, we find the most descriptive images by nearest neighbor search of 32 bit PDH.

Chapter 5

Expanding Visual Categories using Binary Attributes

5.1 Overview

Designing generalizable classifiers for visual categories is an active research area and has led to the development of many sophisticated classifiers in vision and machine learning [93]. Building a good training set with minimal supervision is a core problem in training visual category recognition algorithms [94].

A good training set should span the appearance variability of its category. While the internet provides a nearly boundless set of potentially useful images for training many categories, a challenge is to select the relevant ones – those that help to change the decision boundary of a classifier to be closer to the best achievable. So, given a relatively small initial set of labeled samples from a category, we want to mine a large pool of unlabeled samples to identify *visually different* examples without human intervention.

This problem has been studied by two research communities: active learning and semi-supervised learning. In active learning, the goal is to add visually different samples using human intervention, but to minimize human effort and cost by choosing informative samples for people to label [1, 95, 96]. Even though the amount of human intervention is minimized and its cost is getting cheaper via crowd sourcing, *e.g.*,, Amazon Mechanical Turk, it is still preferable to not have humans in the loop because of issues like quality control and time [95].

Semi-supervised learning (SSL) aims at labeling unlabeled images based on their underlying distribution shared with a few labeled samples [97–99]. In SSL, it is assumed that the unlabeled images that are distributed around the labeled samples are highly likely to be members of the labeled category. However, if we need to dramatically change the decision boundary of a category to achieve good classification performance, it is unlikely that this can be done just by adding samples that are similar in the space in which the original classifier is constructed.

To expand the boundary of a category to an *unseen* region, we propose a method that selects unlabeled samples based on their attributes. The selected unlabeled samples are not always instances from the same category, but they can still improve category recognition accuracy, similar to [100, 101]. We use two types of attributes: category-wide attributes and example-specific attributes. The category-wide attributes find samples that share a large number of discriminative attributes with the preponderance of training data. The example-specific attributes find samples that are highly predictive of the *hard* examples from a category - the ones poorly predicted by a leave one out protocol.

We demonstrate that our augmented training set can significantly improve the recognition accuracy over a very small initial labeled training set, where the unlabeled samples are selected from a very large unlabeled image pool, *e.g.*,, ImageNet. Our contributions are summarized as follows:

1. We show the effectiveness of using attributes learned with auxiliary data to label unlabeled images without annotated attributes.
2. We propose a framework that jointly identifies the unlabeled images and category wide attributes through an optimization that seeks high classification accuracy in both the original feature space and the attribute space.
3. We propose a method to learn example specific attributes with a small sized training set, used with the proposed framework. We then combine the category wide and the example specific attributes to further improve the quality of image selection by diversifying the variations of selected images.

The rest of the chapter is organized as follows: Section 5.2 reviews related work. Section 5.3 presents the overview of our approach. Section 5.4 describes our optimization framework for discovering category wide attributes and the unlabeled images as well as a method to capture exemplar specific attributes. Section 5.5 describes the details of the dataset configurations used in our experiments. Experimental results that demonstrate the effectiveness of our method is presented in Section 10.4.

5.2 Background

Our work is related to active learning, semi-supervised learning, transfer learning and recent work about borrowing examples from other categories.

Active Learning The goal of active learning is to add examples with minimal human supervision [1]. [95] provides a comprehensive survey. Recently, Parakash*et al.* proposed a novel active learning framework based on interactive communication between learners and supervisors (teachers) via attributes [96]. It requires fairly extensive human supervision with rich information.

Semi-Supervised Learning Semi-supervised learning (SSL) adds unlabeled examples to a training set by modeling the distribution of features without supervision. [97] is a detailed review of the SSL literature. Fergus*et al.* proposed a computationally efficient SSL technique for large datasets [98]. Our approach also uses a large dataset and scales linearly in the size of that dataset; it differs from conventional SSL approaches because we do not use the distribution of sample in the original feature space, but in an attribute space. Recently, Shrivastava*et al.* proposed a SSL based scene category recognition framework using attributes, constrained by a category ontology [99]. They leverage the inter-class relationships as constraints for SSL using semantic attributes given by a category ontology as a priori. Our approach is similar to their work in terms of using attributes, but aims to discover attributes without any structured semantic prior.

Transfer Learning and Borrowing Examples Our work is related to recent work on transfer learning [102] and borrowing examples [100, 101, 103].

Ruslan*et al.* [103] proposed building a hierarchical model from categories to borrow images of a useful category for detection and classification. They assume that the images in a category are not diverse and adding all images from some selected category will help to build a better model for the target category. The assumption, however, is bound to be violated by visually diverse categories.

Instead, Lim*et al.* [100] propose a max-margin formulation to borrow some samples from other categories based on a symmetric borrowing constraints.

Kim and Grauman [101] propose a shape sharing method to improve segmentation based on the insight that shapes are often shared between objects of different categories.

Attributes Research on attributes recently has been drawing a lot of attention in the computer vision community because of their robustness to visual variations [104–106]. Attributes can, in principle, be used to construct models of new objects without training data - zero shot learning [105]. Recently, Rastegari *et al.* [107] propose discovering implicit attributes that are not necessarily semantic for category recognition. The discovered attributes preserve category-specific traits as well as their visual similarity by an iterative algorithm that learns discriminative hyperplanes with max-margin and locality sensitive hashing criteria.

5.3 Adding Samples to a Category

Given a handful of labeled training examples per category, it is difficult to build a generalizable visual model of a category even with sophisticated classifiers [93]. To address the lack of variations of the few labeled examples, we expand the visual boundary of a category by adding unlabeled samples based on their attributes. The attribute description allows us to find examples that are visually different but similar in traits or characteristics [104–106].

Based on recent work on automatic discovery of attributes [107] and large scale category-labeled image datasets [108], we discover a rich set of attributes. These attributes are leaned using an auxiliary category-labeled dataset to avoid biasing the attribute models towards the few labeled examples. The motivation here is similar to what underlies the successful Classemes representation [109] which achieved good category recognition performance by representing samples by external data that consists of a large number of samples from various categories.

Across the original visual feature space and the attribute space, we propose a framework that jointly selects the unlabeled images to be assigned to each category and the discriminative attribute representations of the categories based on either a category wide or exemplar based ranking criteria. Sec. 5.4.1 presents the optimization framework for category wide addition of unlabeled samples to categories. This adds samples that share many discriminative attributes amongst themselves and the given labeled training data. The same framework can be applied to identify relevant unlabeled samples based on their attribute similarity to specific instances of the training data. This only involves a simple change to one term of the optimization, and is based on how ranks of unlabeled samples change as labeled samples are left out, one at a time, from the attribute based classifier. So, the optimization runs twice - one to identify samples that share large numbers of discriminative attributes within class and a second to find samples that share strong attribute similarity with specific members of the class, and the two sets of samples are then combined to create the augmented training set for the class. We refer to the first as a categorical analysis and the second as an exemplar analysis.

5.4 Joint Discovery of Discriminative Attributes and Unlabeled Samples

5.4.1 Categorical Analysis

We simultaneously discover discriminative attributes and images from the unlabeled data set in a joint optimization framework formulated in both visual feature space and attribute space with a max margin criterion for discriminativity. Unlike [99], we do not require a label taxonomy to find the shared properties. Also unlike [100], we do not need to learn the distributions of the unlabeled images in the original feature space.

For each category c, we will construct a classifier in visual feature space, w_c^v, using the set $X = \{x_i | i \in \{1, \ldots, l, l+1, \ldots, n\}\}$ that consists of the initially given labeled training images $\{x_i | i \in \{1, \ldots, l\}\} \subset X$ and the selected images from the unlabeled image pool $\{x_i | i \in \{l+1, \ldots, n\}\} \subset X$. The subset of images from the unlabeled set is assigned to a category based on identifying discriminative attribute models. Since the problems of determining the discriminative attributes and selecting the subset of unlabeled data to assign to a category are coupled, we learn them jointly. Additionally, we want to mitigate against unlabeled samples being assigned to multiple categories, so a term $M(\cdot)$ is added to the optimization criteria to enforce that. The joint optimization function is:

$$
\begin{aligned}
&\min_{I_c \in I, w_c^v, w_c^a} \sum_c \Big(\alpha J_c^v(I_c, w_c^v) + \beta J_c^a(I_c, w_c^a)\Big) + M(I) \\
&\text{subject to} \\
&J_c^v(I_c, w_c^v) = \|w_c^v\|_2^2 + \lambda_v \sum_{i=1}^{n} \xi_{c,i} \\
&I_{c,i} \cdot y_{c,i}(w_c^v x_i) \geq 1 - \xi_{c,i}, \quad \forall i \in \{1, \ldots, n\} \\
&J_c^a(I_c, w_c^a) = \|w_c^a\|_2^2 + \lambda_a \sum_{j=1}^{n} \zeta_{c,j} - \sum_{k=l+1}^{n} I_{c,k}\Big(w_c^a \phi(x_k)\Big) \\
&I_{c,j} \cdot y_{c,j}(w_c^a \phi(x_j)) \geq 1 - \zeta_{c,j}, \quad \forall j \in \{1, \ldots, n\} \\
&\sum_{k=l+1}^{n} I_{c,k} \leq \gamma, \quad I_{c,k} = 1, \ \forall k \in \{1, \ldots, l\} \\
&M(I) = \sum\sum_{c1 \neq c2} I_{c1} \cdot I_{c2},
\end{aligned}
\tag{5.1}
$$

$I_c \in \{0, 1\}$ is the sample selection vector for category c, and indicates which unlabeled samples are selected for assignment to the training set of category c. $I_{c,i} = 1$ when the i^{th} sample is selected for category c. $x_i \in \mathbb{R}^D$ is the visual feature vector of image i. $y_{c,i} \in \{+1, -1\}$ indicates whether the label assigned to x_i is c (+1) or not (-1). $\phi(\cdot) : \mathbb{R}^D \rightarrow \mathbb{R}^A$ is a mapping function of visual feature to the attribute space that is learned from auxiliary data, where $\mathbb{R}^D$ and $\mathbb{R}^A$ denote visual feature space and attribute space, respectively. α and β are hyper-parameters for balancing the max margin objective terms for both the visual feature and attribute based classifiers. γ is a hyper-parameter for specifying the number of selected images.

$J_c^v(I_c, w_c^v)$ and the second constraint of Eq. 5.1 are a max-margin classification terms in visual feature space. $J_c^a(\cdot)$ and the forth constraint of Eq. 5.1 are a max-margin classifier in the attribute space (T_A) with a selection criterion (T_R); we divide it as follows:

$$J_c^a(I_c, w_c^a) = \underbrace{\|w_c^a\|_2^2 + \sum_{j=1}^{n} \zeta_{c,j}}_{T_A} \underbrace{- \sum_{k=l+1}^{n} I_{c,k}\Big(w_c^a \phi(x_k)\Big)}_{T_R}. \tag{5.2}$$

T_R essentially chooses the top γ responses of the attribute classifier from the unlabeled set by the fifth constraint of Eq. 5.1. The term $M(I_c)$ penalizes adding the same sample to multiple categories (sixth constraint of Eq. 5.1).

The objective function is obviously not convex due to the interconnection of the two spaces by the example selecting indicator vector I and the attribute mapper $\phi(\cdot)$. However, if the I_c's were known and we fix either $J_c^v(I_c, w_c^v)$ or $J_c^a(I_c, w_c^a)$, the function becomes convex and can be solved with an iterative block coordinate descent algorithm. At each iteration we fix one of the terms and the entire objective function becomes an ordinary max margin classification formulation with a selection criterion. Each iteration of the block coordinate descent algorithm updates the set of indicator vectors I. At the first iteration, the initial value of I is determined by training the attribute classifier w_c^a on the given labeled training set. Then, after the two SVM's in both spaces are updated, we update I. Since there is no proof of convergence for the algorithm, we iterate it a fixed number of times - 1 $\sim$ 5 in practice. The iterations could be controlled using a held out validation set, but since our premise is that labeled samples are rare we do not do that.

5.4.2 Exemplar Analysis

The discriminative attributes learned in Sec. 5.4.1 capture commonality among all examples in a category. We refer them as *categorical attributes*. Each example, however, has its own characteristics that may help to expand the visual space of the category by identifying images based on example-specific characteristics. To discover *exemplar attributes*, a straightforward solution would be to learn exemplar-SVMs [110]. The exemplar-SVM, however, requires many negative samples to make the classifier output stable. For our purposes, though, we can accomplish the same thing by analyzing how the ranks of unlabeled samples change when a single sample is eliminated from the training set of the attribute SVM. If an unlabeled sample sees its rank drop sharply from its rank in the full-sample SVM, then the training sample dropped should have strong attribute similarity to the unlabeled sample.

This is illustrated in Figure 5.1. The top row shows the ten initial labeled orange samples. The leftmost column shows unlabeled samples sorted by their rank in the attribute classifier learned from that set. Then we construct leave one out attribute classifiers, and each column shows the new rankings of unlabeled samples when each image at the top of the column is eliminated from the training set. Eliminating the half orange (second sample, top row) from the training set reduces the rank of the globally best unlabeled sample from 1 to 10.

First, let w_c^a be the attribute classifier for the current training set for category c (while the process is initialized based on the labeled training set, after each iteration we use the additional unlabeled samples added to the category to construct a new attribute classifier). Let $w_{c,\bar{j}}^a$ be the attribute classifier learned when the i^{th} sample is removed from the training set. We next describe how we use the ranks of unlabeled samples in these two classifiers to modify T_R in Eq. 5.2. Basically, we are going to re-rank the unlabeled samples based on their rank changes from w_c^a to $w_{c,\bar{j}}^a$. We want samples whose ranks are lowered dramatically by the elimination of a single sample from the training set to be highly ranked by the re-ranking function. This can be accomplished by computing the following score based on rank changes, and sorting the unlabeled samples by this score:

$$e_j(x_i) = \frac{\mu}{r_g(x_i)} - \frac{\nu}{r_j(x_i)}, \tag{5.3}$$

where x_i is a sample from the an unlabeled pool, $r_g(\cdot)$ and $r_j(\cdot)$ are the rank functions of w_c^a and $w_{c,\bar{j}}^a$ respectively. μ and ν are the balancing hyper-parameters for two ranks. T_R is then simply determined by first selecting the new top ranked sample from each leave one out SVM, then the second ranked, until a fixed number of samples are selected (skipping over duplicates). This set is then used to re-learn the feature and attribute based SVM's and the entire process iterates.

5.5 Dataset

We construct a dataset from a large scale dataset for category recognition, ImageNet [108] using its standard benchmark subset, ILSVRC 2010 dataset. We will publicly release our dataset for future comparison.[1] It consists of approximately 1 million images of 1,000 categories. The images are downloaded from a photo sharing portal[2]. It provides fine grained category labels such as specific breed of dogs, *e.g.,*, Yorkshire Terrier and Australian Terrier.

We randomly choose 11 categories among natural objects such as vegetable and dogs as the categories of interest. Those categories have very large appearance variations due to factors including non-rigid deformation, lighting, camera angle, intra-class appearance variability *etc.*. For each category, we randomly choose ten images as an initial labeled training set and 500 images as a testing set. The unlabeled image pool consists of images that are arbitrarily chosen from the entire 1,000 categories in the ILSVRC 2010 benchmark dataset, but includes at least 50 samples from each of the categories to be learned. The size of the image pool varies in the

[1] http://umiacs.umd.edu/∼jhchoi/addingbyattr/
[2] http://www.flickr.com

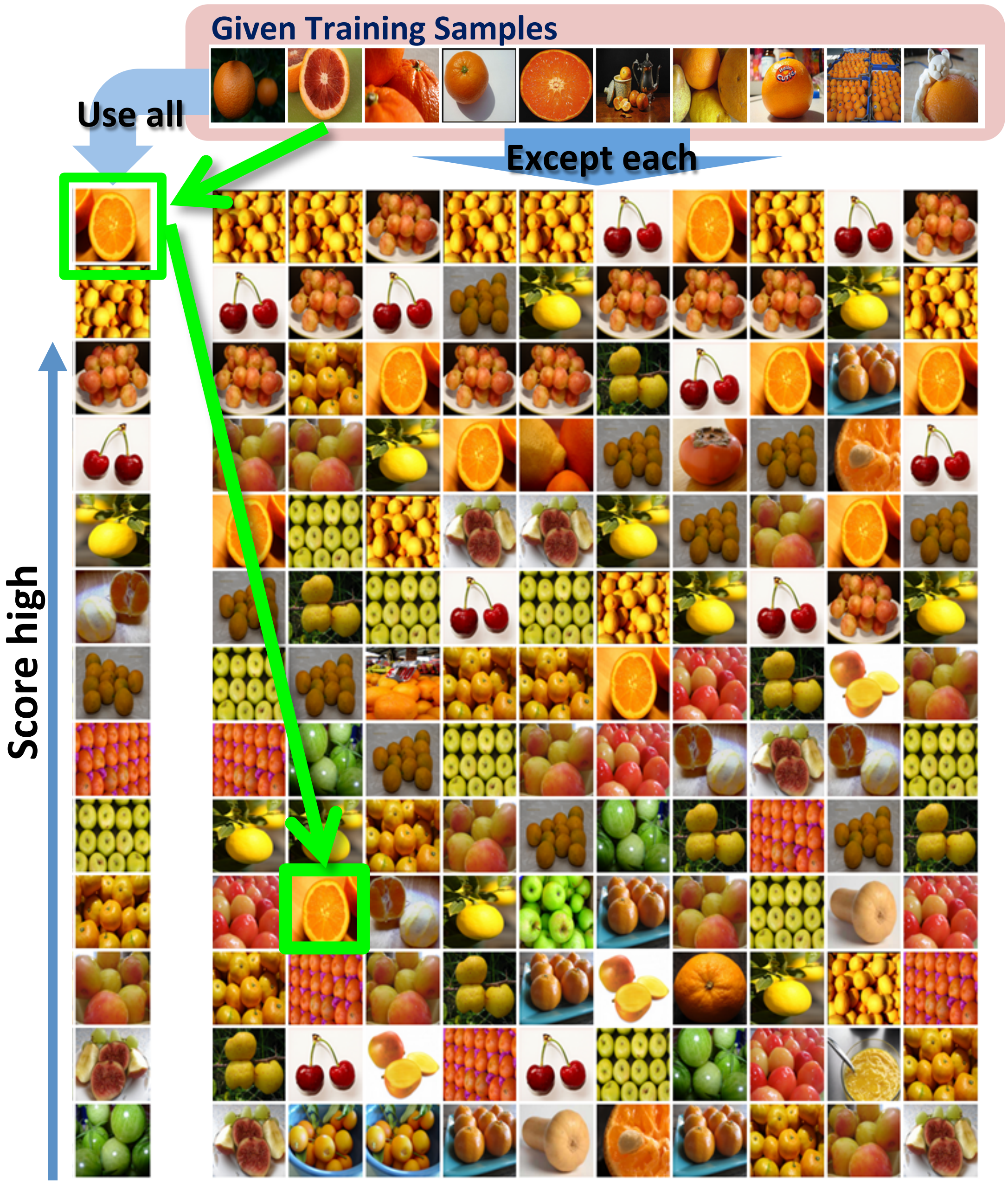


FIGURE 5.1: **Unlabeled images ordered by confidence score by w_c^a and a set of $w_{c,\bar{i}}^a$'s (column wise)**. The first row shows the labeled training samples (10 examples). The left most column is a list of unlabeled images ordered by confidence score by w_c^a. Rest of the columns are lists of unlabeled images ordered by each $w_{c,\bar{i}}^a$'s. Note that an image of halved orange in the second column makes the first ranked images in the left most column (by w_c^a) go down because the halved orange was removed in the training set of $w_{c,\bar{i}}^a$.

experiments but is much larger (from 5,000 to 50,000) than the initial training set. For learning the attribute space and the mapper, it is expected that the attribute mapper should capture some attribute of the categories of interest. For this purpose, we use 50 labeled samples from 93 categories that are similar to the 11 categories to learn the attribute space.

5.6 Experiments

The main goal of our method is to add unlabeled images to the initial training set in order to classify more test images correctly. We demonstrate the effectiveness of our method by improvements in average precision (AP) of category recognition. We also evaluate our approach under various scenarios including the precision (purity) of the unlabeled image pool and the size of the initial labeled set and also the effect of parameters including number of selected examples. Moreover, we evaluate the effect of selecting images that are not from the category of interest.

5.6.1 Experimental Setup

Visual feature descriptors: We use various visual feature descriptors including HOG, GIST and color histograms. Since the feature dimensionality is prohibitively large, we reduce the dimension to 6,416 by PCA.

Attribute discovery: We use the binary attribute discovery method of Rastegari*et al.* [107] as the attribute mapping function, $\phi(\cdot)$ in Eq. 5.1. We learn the mapper with default hyper-parameter sets as suggested in [107]. We use 400 bits in most of our experiments. We also present performance as a function of the number of bits.

Max margin optimization: We use LibLinear [92] for training all max-margin based objective functions. To address the non-linearity of visual feature space, we use homogeneous kernel mapping [111] on the original features with the linear classifier. For the hinge loss penalty hyper-parameter, we use 0.1.

Parameters: For the parameter in Eq. 5.1, we use $\alpha = 1, \beta = 1$. For categorical attribute only, we mostly use $\gamma = 50$ except ones in Section 5.6.4. For combining exemplar and categorical attributes, we mostly use $\gamma = 20$ and $\gamma_i = 3$ except for Section 5.6.4. We investigate algorithm performance as a function of γ in Section 5.6.4. For the parameters of the scoring function for exemplar-attributes in Eq. 5.3, we use $\mu = 1$ and $\nu = 1$.

5.6.2 Qualitative Results

Our method discovers examples that expand the visual coverage of a category by not only adding the examples from the same category but also examples from other categories. Figure 5.2 illustrates qualitative results on the category *Dalmatian* for both categorical and exemplar attributes analyses. The selected examples based on categorical attributes exhibit characteristics commonly found in the labeled examples such as dotted, four legged animal. The exemplar attributes, on the other hand, select examples that exhibit the characteristic of individual labeled training examples.

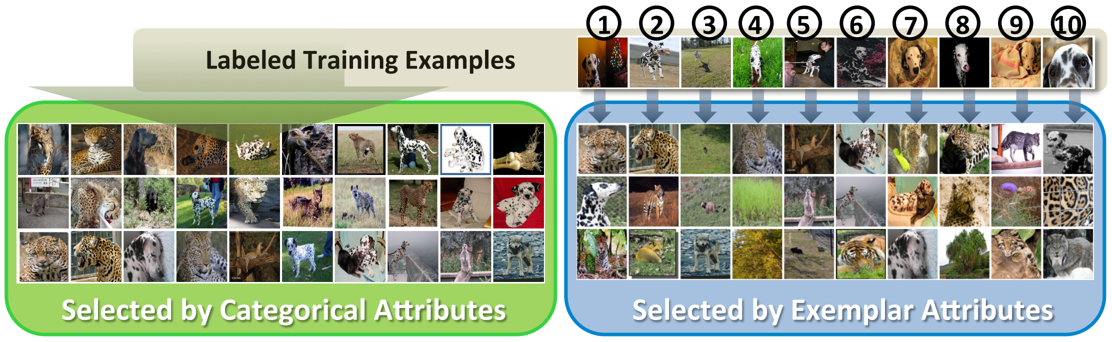

Labeled Training Examples
1
2
3
4
5
6
7
8
9
10
Selected by Categorical Attributes
Selected by Exemplar Attributes

FIGURE 5.2: **Qualitative results of our method**. Note that the selected examples by categorical attributes display characteristics commonly found in the labeled training examples such as 'dotted', 'four legged animal'. In contrast, the exemplar attributes select the examples that display the characteristic of individual example.

5.6.3 Comparison with Other Selection Criteria

Given our goal of selecting examples from a large unlabeled data with only a small number of labeled training samples, we do not compare with semi-supervised learning methods because they need more labeled data to model the distribution. Since our method does not involve human intervention, we do not compare to active learning.

We compare to baseline algorithms which are applicable to the large unlabeled data scenario. The first baseline algorithm is to select nearest neighbors. The second baseline selects images by an active criterion that finds examples close to a learned decision hyperplanes [1]. Both baseline algorithms selects images based on analysis in the visual feature space.

As shown in Table. 5.1, the two baseline strategies decrease mean average precision (mAP). However, our method identifies useful images in the unlabeled image pool and significantly improves mAP by 7.64%. Except for the category *Greyhound*, we obtain performance gain from 2.77% - 16.36% in all categories. The added examples serve not only as positive samples for each category but also as negative samples for other categories. The quality of the selected set can change the mAP significantly in both ways.

5.6.4 Number of Selected Examples

As we select more examples, controlled by γ in Eq. 5.1, the chances of both selecting useful images and harmful images for a category increase simultaneously. We vary the number of selected examples and observe mean average precision as shown in Figure 5.3. The category wide attributes identify useful unlabeled images. In addition, the exemplar attributes further improve the recognition accuracy.

Category Name	Init.	NN	ALC	Cat.	E+C
Mashed Potato	45.03	34.02	51.15	61.39	63.92
Orange	29.84	16.29	26.97	40.61	41.05
Lemon	32.21	27.58	32.43	35.37	34.23
Green Onion	25.06	16.50	19.66	38.57	40.20
Acorn	13.09	11.05	15.41	19.35	20.10
Coffee bean	58.29	43.89	56.62	64.65	66.54
Golden Retriever	14.54	15.57	12.61	17.54	18.61
Yorkshire Terrier	29.62	13.62	27.63	41.41	45.65
Greyhound	15.24	15.73	15.64	14.75	15.22
Dalmatian	43.84	27.97	37.91	54.42	57.23
Miniature Poodle	26.10	12.50	21.16	28.87	30.21
Average	30.26	21.34	28.84	37.90	39.36

TABLE 5.1: **Comparison of average precision (AP) (%) for each category with 50 added examples by various methods**. 'Init.' refers to initial labeled training set. 'NN' refers to addition by 'nearest neighbor' in visual feature space, 'ALC' refers to addition by 'active learning criteria (ALC)' that finds the examples close to the current decision hyperplanes [1]. 'Cat.' refers to our method of select examples using categorical attributes only. 'E+C' refers to addition using categorical and exemplar attributes. The size of the unlabeled dataset is roughly 3,000 from randomly chosen categories out of 1,000 categories.

5.6.5 Adding Examples from Similar Categories

Among the selected images per category, some examples are true instance of the category. We refer to these as *exact examples* and the rest as *similar examples*. We are interested in how much the similar examples improve category recognition. First, we examine the purity of the selected set in Figure 5.4. The purity is the percentage of exact samples in the set. Surprisingly, even though the purity values seem low, they still improve classification performance.

We now investigate how much the similar examples improve the average precision (AP) by removing the exact examples from the selected set. The blue bars in Figure 5.5 represent the AP using just the similar examples. It is interesting to note that using only the similar examples still improves the APs over the initial labeled set.

In addition, it is also interesting to observe how the performance changes when we add the same number of similar examples as the size of the initially selected image set (50). This is shown as green bars in Figure 5.5. All results are obtained using categorical attributes only. (The results using both exemplar and categorical attributes are similar so are omitted).

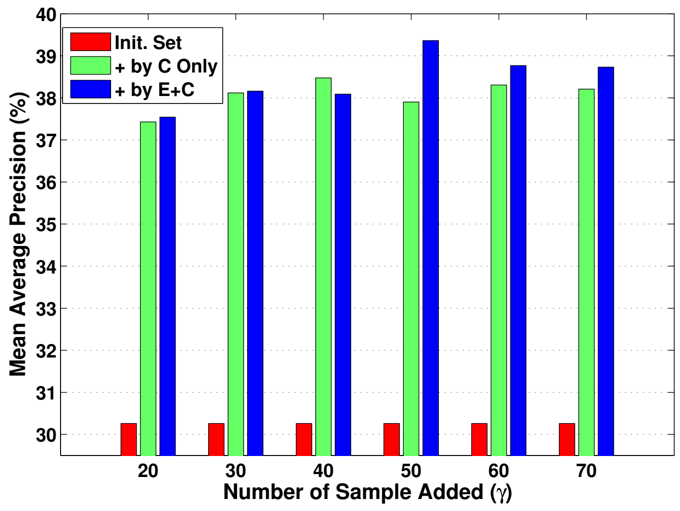


FIGURE 5.3: **Mean average precision (mAP) of 11 category by our method varying the number of unlabeled images selected**. The red, green and blue are the mAP using the initial labeled set (Init. Set), the augmented set by our method using category wide attributes only (+ by C only) and categorical+exemplar attributes respectively. (+ by E+C)

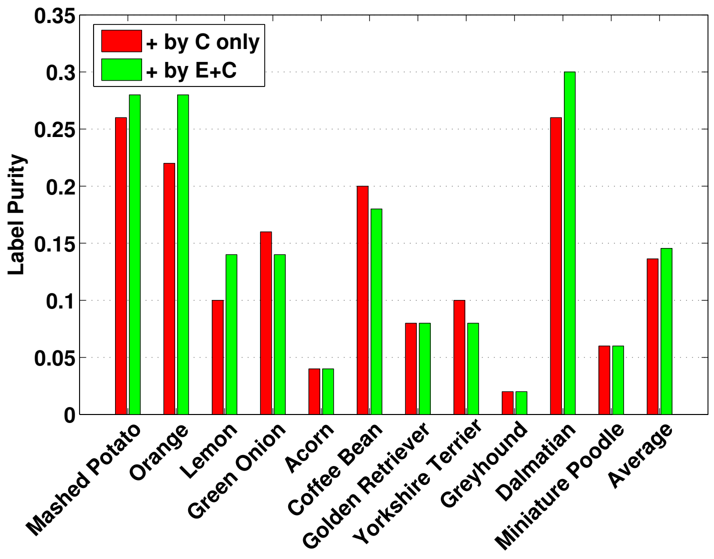


FIGURE 5.4: **Purity of added examples**. Red bars denote the purity of selected images using category wide attributes only (+ by C only) and the green bars are obtained from categorical+exemplar attributes (+ by E+C).

5.6.6 Precision of Unlabeled Data

The unlabeled data can be composed of images from many categories. The precision of the unlabeled data is defined as the ratio of size of the unlabeled images from extraneous categories to the size of the entire unlabeled image data. The larger the unlabeled data, the lower we expect its precision to be (imagine running a text based image search using the category name and

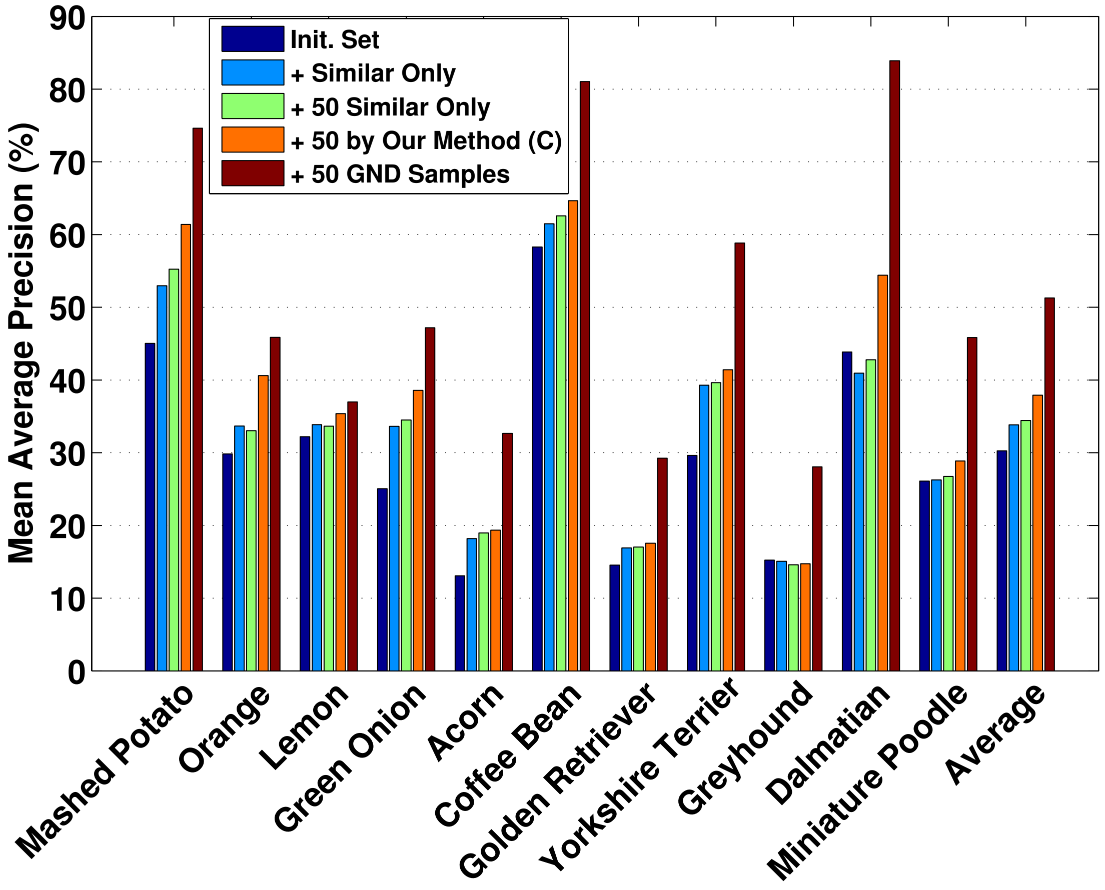


FIGURE 5.5: **Mean average precision (mAP) as a function of the purity of the selected examples**. The navy colored bars are obtained using the initial labeled set (baseline). The blue bars use only similar examples among the selected 50 examples. The green bars use 50 similar examples to compare with the result of our selected 50 examples (orange bars) including both similar and exact examples. The red bars are obtained using a set of 50 ground truth images, which is the best achievable accuracy (upper bound). Even the similar examples alone improve the category recognition accuracy compared to just using the initial labeled set.

accepting the first k images returned). It is interesting to observe how robust our method is against the precision of unlabeled data.

We start with an unlabeled set (550 images, 50 from each of the 11 categories) of precision 1.0, and reduce precision by adding images from other categories. The number of the unrelated images ranges from 2,500 to 50,000, which are randomly chosen from the entire 1,000 categories of the ImageNet ILSVRC 2010 dataset.

As shown in Figure 5.6, we observe that the accuracy improvement by our method using categorical attributes is quite stable even when precision is low.

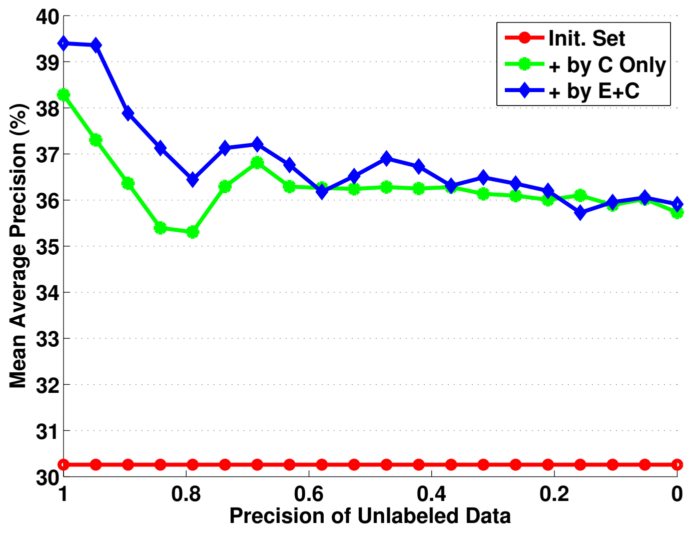


FIGURE 5.6: **Mean average precision (mAP) as a function of precision of unlabeled data**. Precision denotes the ratio of size of the unlabeled images from extraneous categories to the size of the entire unlabeled image data (size = 50,000). Although precision decreases, the mean average precisions (mAP) by our method do not decrease much.

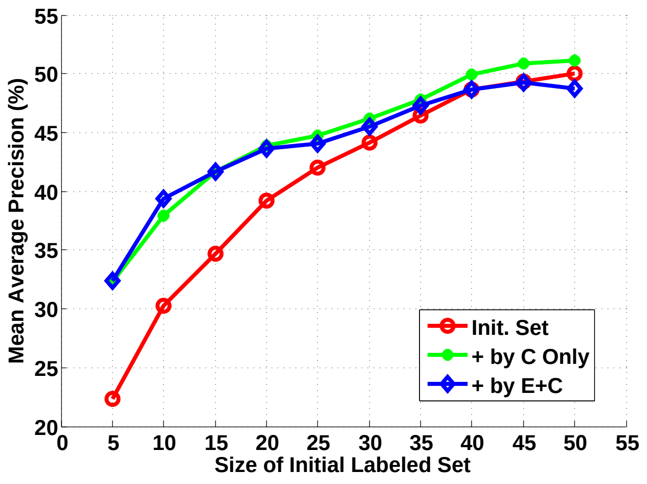


FIGURE 5.7: **Mean average precision (mAP) as a function of the size of the initial labeled set**. The number of added samples is 50 in all experiments.

5.6.7 Size of Initial Labeled Set

We next explore how the size of the initial labeled set effects accuracy. We systematically vary the size from 5 to 50 and show mAP compared to an SVM learned on the initial training set - see Figure 5.7. The mAP gain for the smallest initial labeled set (5) is the highest as expected. When the number of samples is larger than 25, our method (+ by C only) does not improve the mAP much, although it still improves by $1.18 - 2.74\%$. Interestingly when there are many samples in the initial training set (*e.g.*,, more than 25), the exemplar traits begin to reduce the mAP.

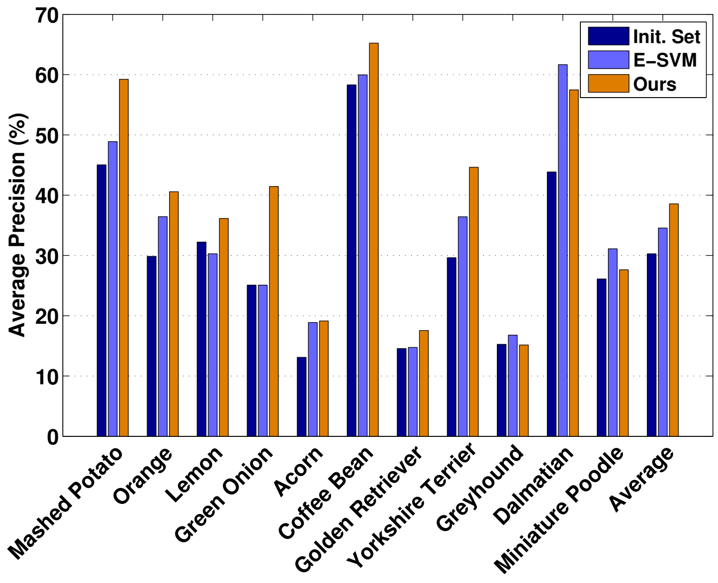


FIGURE 5.8: **Comparison of our exemplar attribute discovery method (Sec. 5.4.2) to exemplar SVM**. Our method outperforms the exemplar SVM in terms of category recognition accuracy by APs without the extra large negative example set (size = 50,000).

5.6.8 Comparison to Exemplar SVM

We also compare the effectiveness of our proposed exemplar attributes discovery method (Sec. 5.4.2) to a conventional exemplar SVM [110]. It is straightforward to integrate the exemplar SVM into our formulation (Eq. 5.1): by setting label $y_{c,j}$ to 1 for the j^{th} example, the label coresponding to the examples in the same category to 0 and the rest to 1. To stabilize the exemplar SVM scores, we employ 50,000 external negative samples to learn each exemplar SVM while we use the small original training set for our method. Figure 5.8 shows that our exemplar attribute discovery method outperforms the exemplar SVM by large margins even without the large negative example set.

Chapter 6

Multi-Attribute Queries: To Merge or Not To Merge?

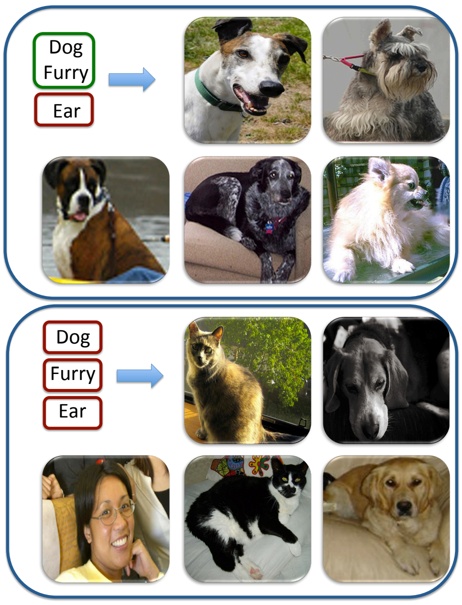


FIGURE 6.1: In a multi-attribute image search, some combinations of attributes can be learned jointly, resulting in a better classifier. In this paper, we propose a model to predict which combinations will result in a better classifier without having to train a classifier for all possible cases. For example, when looking for dog, furry, ear, our method selects to train a furry-dog classifier and fuse it with an ear classifier. We compare this selection with the default case where one classifier is trained per attribute. Here we show top five retrieved images.

6.1 Overview

We often find ourselves searching for images with very specific visual content. For instance, if we witness a crime we might help law enforcement agents search through mugshots of criminals to find the specific individual we saw. Victims of disasters may search through hospital databases to find missing loved ones. Graphic designers may search for illustrations of specific styles. Bird watchers may search for photographs of birds with a particular appearance to identify its species. In such scenarios, the most natural way for users to communicate their target visual content is to describe it in terms of its attributes [104, 105] or visual properties. Given the specificity of the desired content, the user typically needs to specify multiple attributes in order to appropriately narrow the search results down.

A common way of dealing with such multi-attribute queries is to train classifiers for each of the attributes individually and combine their scores to identify images that satisfy all specified attributes. If a user is interested in images of white furry dogs, one would run three classifiers and combine them (white & furry & dog) to indirectly get a white-furry-dog classifier. However this may not be the most effective or most efficient solution. White furry dogs may have a very characteristic easy-to-detect appearance, and running just one white-furry-dog classifier trained to directly detect only white furry dogs could result in more accurate and faster results. But there may not be enough white furry dog examples to train such a classifier. Or, white furry dogs may look a lot like the rest of the dogs leading to a harder classification problem and poorer performance than combining three independent classifiers. Given a multi-attribute query such as white furry dog, it is critical to determine which combinations of classifiers should be trained to ensure effective and efficient retrieval results: white-furry & dog, or white-furry-dog, or white & furry & dog, etc.

An exhaustive solution to this problem would involve training all possible combinations of the multiple attributes involved (5 combination in the case of white furry dogs), and evaluating their accuracy on a held out set of images to determine the optimal combination. This would be computationally expensive especially as the number of attributes in the query grows, and requires sufficient amount of validation data. We propose an optimization approach that given a multi-attribute query, efficiently identifies which components would be beneficial *i.e.,*which attributes should be merged, without having to enumerate and train all possible combinations. We use the intuition that geometric notions that capture the compactness ($\sim$intra-class variance) of the set of images that satisfy a combination (*e.g.,*white-dog), and the margin of these images from other distractor images ($\sim$inter-class variance) provide good proxies for the likely effectiveness of a classifier trained to recognize the combination. We show that these geometric quantities can be evaluated efficiently in a discriminative binary space. We evaluate our algorithm on aPascal and Bird200 datasets and show that our method can find combinations that are both more accurate and faster than independent classifiers.

6.2 Background

We now describe the connections of our work to existing work on dealing with multi-attribute queries and visual phrases. We also briefly mention other uses of binary spaces in literature.

Multi-attribute queries: Attributes or semantic concepts are often used for improved multimedia retrieval [112–118]. Fewer works have looked at the challenges that arise in multi-attribute queries in particular. Siddiquie *et al.* [119] model the natural correlation between attributes to improve search results. For instance, if a face has a mustache then it is likely to be a male face. Scheirer *et al.* [120] recently proposed a novel calibration method to more effectively combine scores of independent multiple attribute classifiers. Our work is orthogonal to these efforts. We are interested in identifying which attributes should be merged to then train a classifier directly for the conjunction for improved search results. Note that we identify beneficial conjunctions for each given multi-attribute query, and do not reason about global statistics of pre-trained attribute classifiers.

Visual phrases: The attribute combinations we reason about can be thought of as being analogous to the notion of visual phrases introduced by Sadeghi *et al.* [121]. They showed that some *object* combinations correspond to a very characteristic appearance that makes detecting them as one entity much easier. For instance, one can detect a person riding a horse more accurately if modeled as one entity, than detecting the person and horse independently and then combining their responses. They used a pre-defined vocabulary of visual phrases. Our work is distinct in that it deals with attribute combinations rather than object compositions. More importantly, the goal of our work is to identify which combinations should be trained on a *per query basis*. Li *et al.* [122] proposed an approach to identify which groups of objects should be modeled together. They reason about consistent spatial arrangements of objects in images. This would be analogous to reasoning about ground truth attribute co-occurrence patterns when dealing with multi-attribute queries. In contrast, in our work we explicitly reason about the variation in appearances of images under the different attribute combinations. As a result, the combinations we identify are grounded to the appearance features of images, which significantly affect the accuracy of resultant classifiers.

Binary spaces: There has been significant progress in recent years in mapping images to binary spaces. One might learn a mapping that preserves correlations between semantic similarities and binary codes [**?**], or local similarities [38, 51, 123]. Recently, discriminative binary codes have shown promising results in mapping images to a binary space where linear classifiers can perform even better than sophisticated models [124]. We use this mapping to project images to a binary space where computing simple geometric measures like compactness or diameters of a group of images and their margins from other images is very efficient.

6.3 Finding Learnable Components

Given a multi-attribute query, our goal is to figure out which combinations of attributes would be better to use without having to train classifiers for all possible combinations. What makes a combination desirable? The most important criteria is the *learnability* of a combination. In other words, we should learn a classifier for a combination of two attributes if it results in a better classifier for the conjunction than combining scores of independent attribute classifiers post-training. For three attributes like white and furry and dog[1], a combination can include multiple components like white and furry-dog. We argue that geometric reasoning in terms of the tightness and margin of each component in a combination is a reasonable proxy for what would have happened if we would have trained a classifier for each component in the combination. Geometrically speaking, a good combination should have components that occupy tight regions of the feature space and have large margins. Figure 6.2 shows an illustration where purple instances are the ones that have both blue and red attributes. What justifies learning a red-blue classifier instead of red and blue classifiers independently is that purple instances occupy a tight area in the feature space with big margins from other blue and red instances. If it was not the case, then we could have learned separate red and blue classifiers; they are more widely applicable and would not sacrifice training data. To efficiently compute these geometric measurements we propose to map the images from the original feature space into a binary space where discriminative properties are preserved. In this section we assume that such a mapping exists. Later in the experiments we show that our formulation is not very sensitive to the choice of the mapping as long as discriminative properties are preserved/enhanced in the binary space. This is not a restrictive condition as most existing binary mapping approaches in literature meet this criteria.

We estimate the learnability of a combination based on the diameter of the components in the combination and the margin within and across components. To setup notations, let's assume there are n attributes involved in a given multi-attribute query, $A = \{a_1, ..., a_n\}$. For example, {white,furry,dog}. There are 2^n different ways to form components. For instance, {white}, {furry,dog}, {white,dog}, {furry}, etc. The set of all possible components is the powerset of A, which we call $\mathcal{S} = \{S_1, S_2, \cdot, S_m\}, m = 2^n$. A combination is a subset of $\mathcal{S}$ that covers A *e.g.,*{{white,furry}, {dog}}, which we write as {white-furry,dog} in shorthand. We define the learnability of a combination $\mathcal{C}$ as

$$\mathcal{L}(\mathcal{C}) = \sum_{c \in \mathcal{C}} [\sum_{c' \in \mathcal{C}, c' \neq c} \mathcal{K}(c, c') + \sum_{a \in c} \mathcal{K}(c, c \setminus a) - \mathcal{D}(c)]$$

where c indexes components in the combination $\mathcal{C}$, a indexes attributes in each component, $\mathcal{D}(c)$ is the diameter of each component defined as $\max_{x,y \in c} d(x, y)$ where x and y are images that

[1]For generality of discussion, we treat all words involved in a query as "attributes"

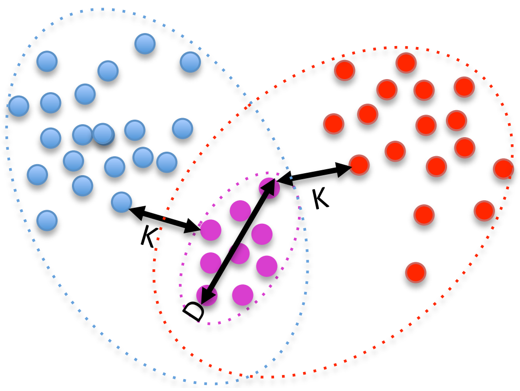


FIGURE 6.2: What makes merging two attributes desirable? When instances that satisfy both attributes occupy a tight region in the feature space and have enough margin to the instances that have one of the attributes. This figure depicts a case where training a merged red-blue classifier is beneficial. Because purple dots (instances that have both red and blue attributes) have small diameter (D) and enough margins (K) with the rest of blue and red dots.

belong to a component and d is the distance between them. The diameter captures the range of visual appearances of images within a component. The higher the variety of appearances, the less learnable the corresponding component. $\mathcal{K}(c, c')$ is the margin between two components c and c' defined as $\min_{x \in c, y \in c'} d(x, y)$. This captures how distant the images belonging a component are from images of other components. The more distant they are, the easier it is to learn a classifier for the component. Finally, $\mathcal{K}(c, c \setminus a)$ is the margin between images that satisfy all attributes of a component, and those that satisfy all but one attribute. For example the margin between purple and red in Figure 6.2. For components that consist of only one attributes the within component margins are zero.

We are interested in finding the optimal combination $\mathcal{C}^*$ that obtains best learnability score and covers all members of A without being inefficiently redundant . We can formulate this problem as the following integer program:

$$\begin{aligned} \max_{x} \quad & \mathcal{L}(\mathcal{S} \odot x) - \lambda |x| \\ & Z^T x \geq \mathbf{1} \\ & x \in \{0, 1\}^m \end{aligned} \tag{6.1}$$

where $\odot$ is the set selection operator, Z is an $m \times n$ binary set system matrix indicating which attributes appear in which component, λ is the trade off factor between the number of components in a combination (efficiency) and the learnability score, and x is the indicator vector that identifies which components will make it to the final combination.

Set covering problem can be reduced to our problem. The optimization 6.1 is harder than standard weighted set covering problem because our learnability function $\mathcal{L}$ defines over all component in a combination. The corresponding weighted set cover formulation requires the weighting function to be defined over each component independently. The interdependencies between components in our learnability function make this optimization NP-hard. However, our learnability function doesn't face an interdependency issue in case of two attributes. This suggests defining a gain function for pairs of attributes that takes into account the same measurements (diameter and margins) as in our learnability function:

$$\mathcal{G}(a_i, a_j) = \mathcal{K}(a_i a_j, a_i) + \mathcal{K}(a_i a_j, a_j) - \mathcal{D}(a_i a_j)$$

Given two attributes, positive values for the gain function recommend merging the two attributes and negative values encourage training separate classifiers for each attribute and then merging their scores. The higher the gain function the higher is the reward for merging two attributes. Our gain function exposes an interesting property that helps prune the search space drastically.

Lemma 6.1. *If attributes a_i and a_j are merged because $\mathcal{G}(a_i, a_j) \geq 0$ then for any other attribute a_k, $\mathcal{G}(a_i a_j, a_k) \geq \mathcal{G}(a_i, a_k)$ or $\mathcal{G}(a_j, a_k)$*

Proof. It's simple to show that if $A \subset B$ then $\mathcal{D}(A) \leq \mathcal{D}(B)$, and if $C \subset D$ then $\mathcal{K}(A, C) \geq \mathcal{K}(B, D)$. We can show that $\mathcal{G}(a_i a_j, a_k) = \mathcal{K}(a_i a_j a_k, a_i a_j) + \mathcal{K}(a_i a_j a_k, a_k) - \mathcal{D}(a_i a_j a_k) + > \mathcal{K}(a_i a_j a_k, a_i a_j) + \mathcal{K}(a_i a_j a_k, a_k) - \mathcal{D}(a_i a_k) + > \mathcal{K}(a_i a_k, a_i) + \mathcal{K}(a_i a_k, a_k) - \mathcal{D}(a_i a_k) + = \mathcal{G}(a_i, a_k)$. The same holds for $\mathcal{G}(a_i, a_j)$. $\square$

What this lemma implies is that once two attributes are merged, we need not consider merging any other attribute with either of these attributes individually. This suggests the following recursive greedy solution to find the highest scoring and covering combination.

Our greedy solution starts with computing the gain for all pairs of attributes. It picks the pair with the highest gain. If the highest gain is positive, then we merge those attributes and add a new merged-attribute to our set of attributes and remove the two independent ones. Meaning that if a_i and a_j provide the biggest positive gain we add $a_i a_j$ as a new attribute to A and remove a_i and a_j from the set. The Lemma above shows that it is safe to remove the independent attribute from the set as no other attribute can join either of a_i or a_j independently and result in higher scoring combination. The new A now has $n - 1$ elements. We can recursively repeat this procedure till we cover all attributes. If there is no pair with positive gain, we move to triplets. This never happened in our experiments.

Efficient Computation of Geometric Measurements: Margins and diameters can be computed efficiently in a binary feature space; $O(NK)$ where N is the number of images and k is the dimensionality of bit vectors. The core part for computing both margin and diameter is to compute the average of all pairwise distances. A naive algorithm would be to go over all pairs and compute their distances and get mean of them. But since we are using binary codes for each dimension of the binary codes we can compute number of **zero** bits and number of **one** bits. Then the sum of the distance of any given bit to all other bits can be computed in $O(constant)$. Algorithm 1 explains this algorithm more formally.

Algorithm 4 Efficient Sum of Pairwise Hamming Distances

```
Input: $B1$ , $B2$ are a binary matrix of size $N \times K$.
Output: $S$: sum of hamming distances between all pairs of rows in $B1$ and $B2$.
 1: for $k = 1 \to K$ do
 2:     $Z(k) \leftarrow \sum_k B2(:,k)$ Comment: Counting Number of zeros in $k^{th}$ dimension of $B2$
 3:     $O(k) \leftarrow \sum_k \neg B2(:,k)$ Comment: Counting Number of ones in $k^{th}$ dimension of $B2$
 4: for $i = 1 \to N$ do
 5:     for $k = 1 \to K$ do
 6:         if $B1(i,j) = 0$ then
 7:             $P(i,j) \leftarrow O(k)$
 8:         else
 9:             $P(i,j) \leftarrow Z(k)$
10: $S \leftarrow \sum P$ Comment: Sum of all elements in $P$
```

6.4 Experiments

We evaluate our method in several different settings. We conduct experiments on two challenging datasets: the aPasclal [104] and the Caltech Bird200 dataset [125]. We compare our method with four different baselines described later. We also test our method with different binary code mapping methods and show that our method is robust to the choice of binary mapping. We also evaluate the impact of different binary code sizes on the performance of our approach. In addition to accuracy, we also compare the running time of our method to that of baselines. We find that our low complexity $O(NK)$ gives us one order of magnitude speed up. We also present qualitative results and analysis that reveal the tendencies of different attributes to merge with other attributes.

6.4.1 Datasets

aPASCAL [104]: This dataset contains the 20 PASCAL object categories. On average each category has 317 images. Each image is labeled by 64 attributes that describe different object properties such having a particular body part, types of materials, etc. We experiment with the low-level features provided by the author of [104] on the data set website and also train/test splits provided with the dataset. The features and attribute annotations are not labeled for entire image. They are computed only for bounding box of the objects.

Caltech-UCSD Bird200 [126] This data set is a challenging subordinate recognition dataset. It includes 200 different species of North American birds with on average 300 images per category. Each image is annotated with 312 bird attributes such as color and shapes of wings, beaks, etc. We used the low-level features provided by [127] describing color, shape and contours. Similar to aPascal, here, we don't use entire image, we ony use the area that the bounding box of the image specifies for a bird in that image. We devide each category in half and took one haf as train set and the other half as test set.

6.4.2 Baseline Methods

We compare our method with four different baseline approaches for selecting the combinations to be trained for a given multi-attribute query: **Default (DEF):** As the name suggestions, this approach uses the most natural strategy of training classifiers for each of the attribute independently and then combining the result scores. **Random Selection (RND):** This approach randomly selects a combination from all possible combinations and learns a classifier for each component of that combination. **Upper Bound (UPD):** Here we exhaustively train all possible combinations, evaluate their performance *on the test set*, and select the best one. The resultant performance corresponds to the upper bound one can hope to achieve by picking the optimal combinations to train. Of course, our proposed approach avoids training all possible combinations, and selects a good combination very efficiently. A comparison to this upper bound informs us of the resultant loss in performance by trading it off for efficiency. **Best Attribute First (BAF):** Intuitively, if an attribute predictor is accurate enough (in the limit, perfect), there is no benefit to merging it with another attribute. This baseline is based on this intuition. It determines which attributes to merge by looking at their prediction accuracies on the test set. Attributes with an accuracy higher than a threshold are left alone, while the rest are merged. We search for a threshold that gives us highest overall accuracy on all the queries.

6.4.3 Evaluation

Having identified the best combination (*e.g.,*{white-furry,dog}), we train a classifier for each of the components {white,furry} and {dog} using (with C = 1). All training images that are both white and furry are positive examples to train a white-furry component classifier, and all remaining images are negative examples. Given a test image, we compute its score for each of the component classifiers. A naive way of combining these component classifiers would be to threshold the scores and compute a logical-AND. However in practice, the scores of the different classifiers are not calibrated. We use [128] to calibrate the scores, which fits a weibull distribution to the scores of a classifier to generate probability estimates. We later show the benefits of this calibration. We threshold the calibrated probabilities and compute the logical-AND to

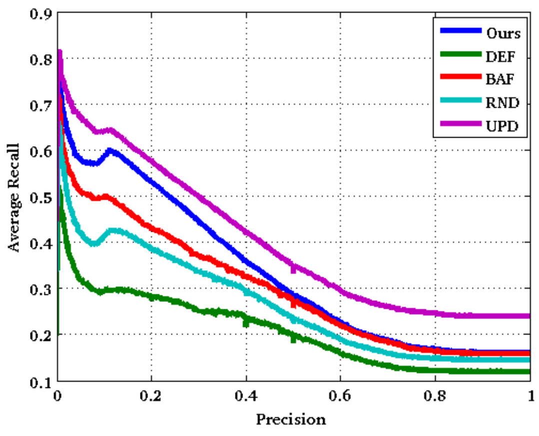


FIGURE 6.3: We evaluate our method on retrieving images in aPascal test set using 3-attribute queries. We compare it with three baselines and also the best possible upper bound. We use 512-dimensional bit codes for this experiment. Each point in this plot corresponds to average recalls over selected combinations on several fixed precisions. The threshold for BAF is 0.7.

determine if a test image is positive (relevant to the multi-attribute query) or not. Varying the threshold gives us a precision-recall curve. One might argue that by taking the product of the calibrated scores and then thresholding that we may get better performance. But in our experments it drops the performace remarkably. In order to report results across multiple queries, we average the recall across all queries for fixed precision values to obtain an "average" precision-recall curve.

Comparison with Baselines: We generated 500 random 3-attribute queries that had atleast 100 corresponding images in the train and test splits. We also generated another set of 500 3-attribute queries that had between 5 and 50 examples in the train and test splits. This allows us to evaluate our approach on queries with sufficient as well as few examples. Figure 6.3 shows our results for the aPascal. We see that our method outperforms all baselines, and is not significantly worse than the upper-bound, especially at high recall. For these experiments we used 512 bits codes extracted using Discriminative Binary Codes [124]. Figure 6.4 shows results using 4-attribute queries, with similar trends. Figure 6.5 shows our results on the Birds dataset with queries of length 3. Our method outperforms the baselines by large margin. The effects of different parts in learnability function at 0.2 precision is as follow: Recall .15 .45 .61. $\mathcal{K}(c, c')$: 102 170 213. $\mathcal{K}(c, c \setminus a)$: 23 56 79. $\mathcal{D}(c)$: 162 106 62. Increase in the margin and decrease in the diameter results in better recall.

Binary Code Length: We now investigate the effect of different length of binary codes on the performance of our method. Figure 6.6 shows results aPascal using the same length 3 queries described earlier. Using fewer bits hurts performance. Figure 6.7 shows similar trends on the Birds dataset.

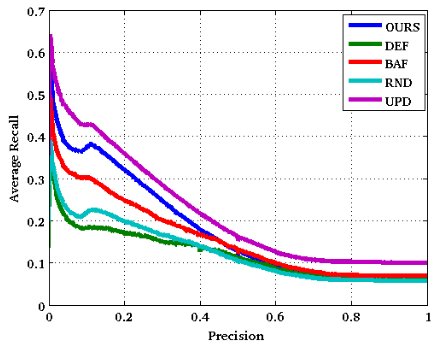


FIGURE 6.4: We evaluate our method on retrieving images in aPascal test set using 4-attribute queries. Experimental setup is similar to that of Figure 3. The threshold for BAF is 0.82 .

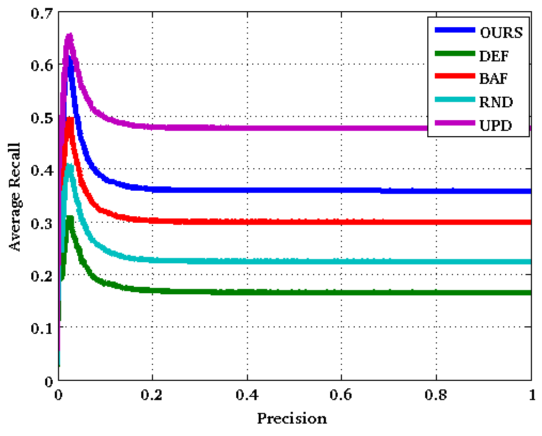


FIGURE 6.5: We evaluate our method on retrieving images in Bird test set using 3-attribute queries. Experimental setup is similar to that of Figure 3.

Sensitivity to Binary Mapping Methods: We now evaluate our model using binary codes generated by different methods. We chose two state-of-the-art binary mapping methods DBC [124] and ITQ [51] and also classical LSH [38]. Table 6.1 compares the performance of our approach using these three methods on the aPascal dataset. Here we use mean of the average recalls over all fixed precisions (MAR) as a measure for comparison. We used 512 bits for all of the methods. DBC perform slightly better because DBC preserves categorical similarities between images. We trained DBC on the whole train set of aPascal dataset. To make the most of ITQ we used the attribute labels of the train set to learn ITQ coupled with CCA. The binary codes produced by ITQ-CCA are expected to preserve pairwise similarities. For both cases we use their publicly available MATLAB code. Our model is not sensitive to the choice of binary mapping (compare DBC and ITQ) as long as discriminative properties can be preserved.

Running Time Evaluation: Here we report the run time of our approach. First, we only consider the average time required to find the best combination for a given query. Table 6.2 compares our method with UPD on 1000 queries of length 3 on the aPascal dataset. Our method

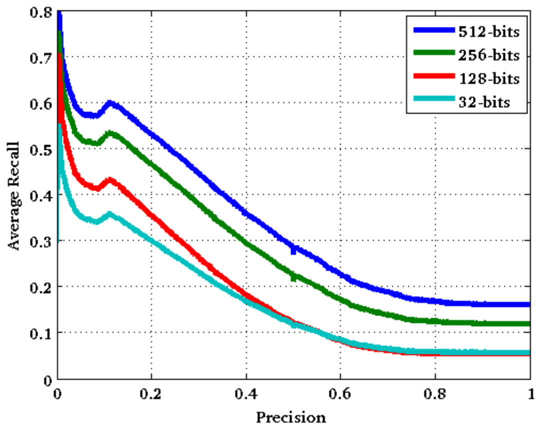


FIGURE 6.6: We investigate the effects of the dimensionality of binary space on our performance on the aPascal dataset.

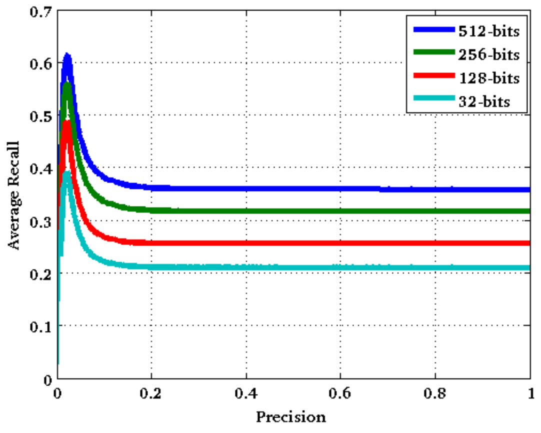


FIGURE 6.7: We investigate the effects of the dimensionality of binary space on our performance on the Bird dataset.

Method	MAR
Upper Bound	0.4007
DBC-512bits	0.3348
ITQ-CCA-512bits	0.3257
LSH-512bits	0.3071

TABLE 6.1: Comparison between different binary mapping methods in terms of Mean Average Recall.

is one order of magnitude faster than UPD which verifies that our algorithm for computing the sum of pairwise distance in the binary space is very fast and efficient. Second, we consider the entire retrieval task which involves identifying the best combination, learning the corresponding component classifiers and finally evaluating them on test images. Table 6.3 compares our model with UPD and DEF. Interestingly, our method is also faster than DEF. This is because in DEF

Method	Time(Second)
Upper Bound	35.325
Ours	0.508

TABLE 6.2: Time for finding best combination: Trying all possible combinations of attributes and picking the best one is very expensive. This table compares the time needed to compute the upper bound versus the time that our algorithm needs to decide which combination to pick.

Method	Time(Second)
Upper Bound	167.68
Default	42.56
Ours	22.34

TABLE 6.3: Average Retrieval Time : Comparisons between the entire time needed to perform the default case, our method, and the upper bound. This table assumes that no classifiers for the default case are trained off line.

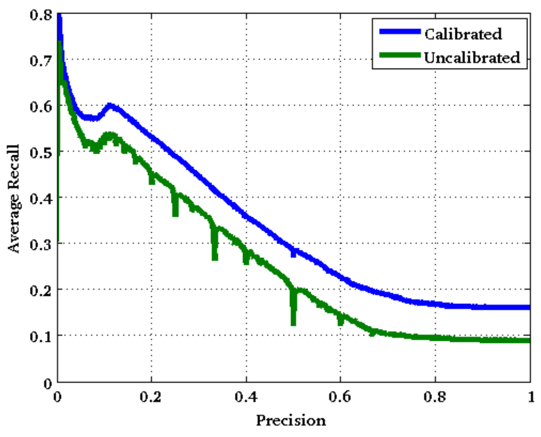


FIGURE 6.8: Calibration Effects

we always need to train n(n: query length) classifiers but in our model on average we need to learn 1.4 classifiers. This comparison assumes that no computations are being done off line. One advantage of DEF over our method is that training and testing in DEF can be done off line.

Calibration Effect: As discussed earlier, calibration is very important when combining multiple component classifiers. Figure 6.8 empirically verifies this by comparing the performance of UPD with and without calibration on the aPascal data set. Without calibration the performance is almost 5% worse.

Qualitative Evaluation: Finally, we look at some qualitative retrieval results comparing our approach to DEF and UPD. Figure 6.10 presents top five images retrieved by different methods for several multi-attribute queries.

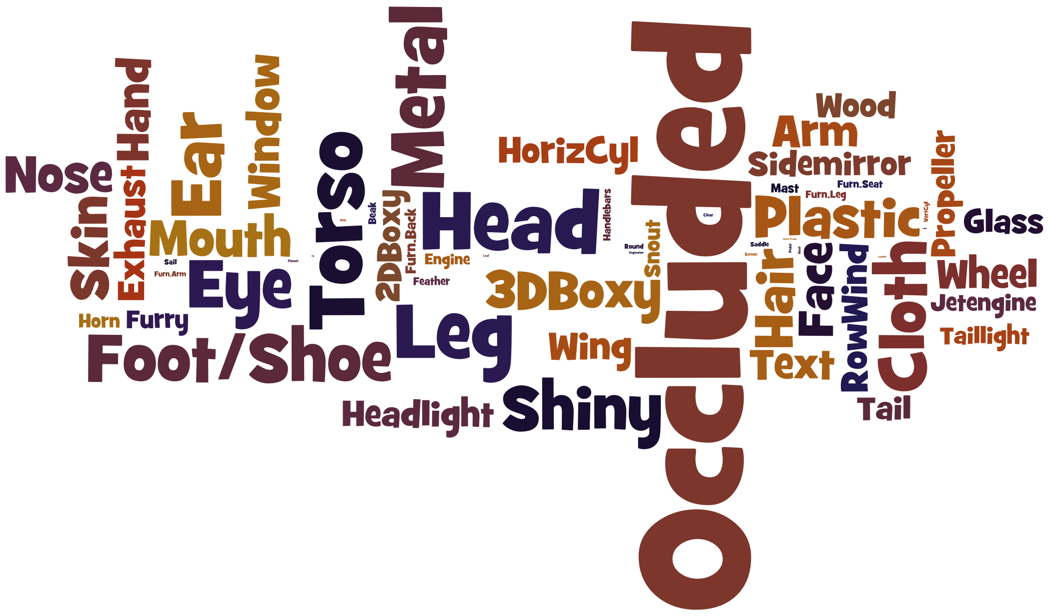


FIGURE 6.9: Some attributes have the tendency to be merged and some prefer to stay separated. The bigger the names in this figure the higher the tendency of the attribute to merge. It is interesting to see that attributes like occluded tend to merge frequently. This is probably because of the fact that the appearance of attributes like this varies a lot as they appear with other attributes. On the other side, attributes like beak and furniture leg tend to be separated as their appearance does not change in combinations.

We now look at which attributes tend to merge with other attributes often, and which ones typically stay un-merged. We created a wordle using wordle.net as seen in Figure 6.9. The bigger the font size of a word, more likely is the corresponding attribute to merge with other attributes.

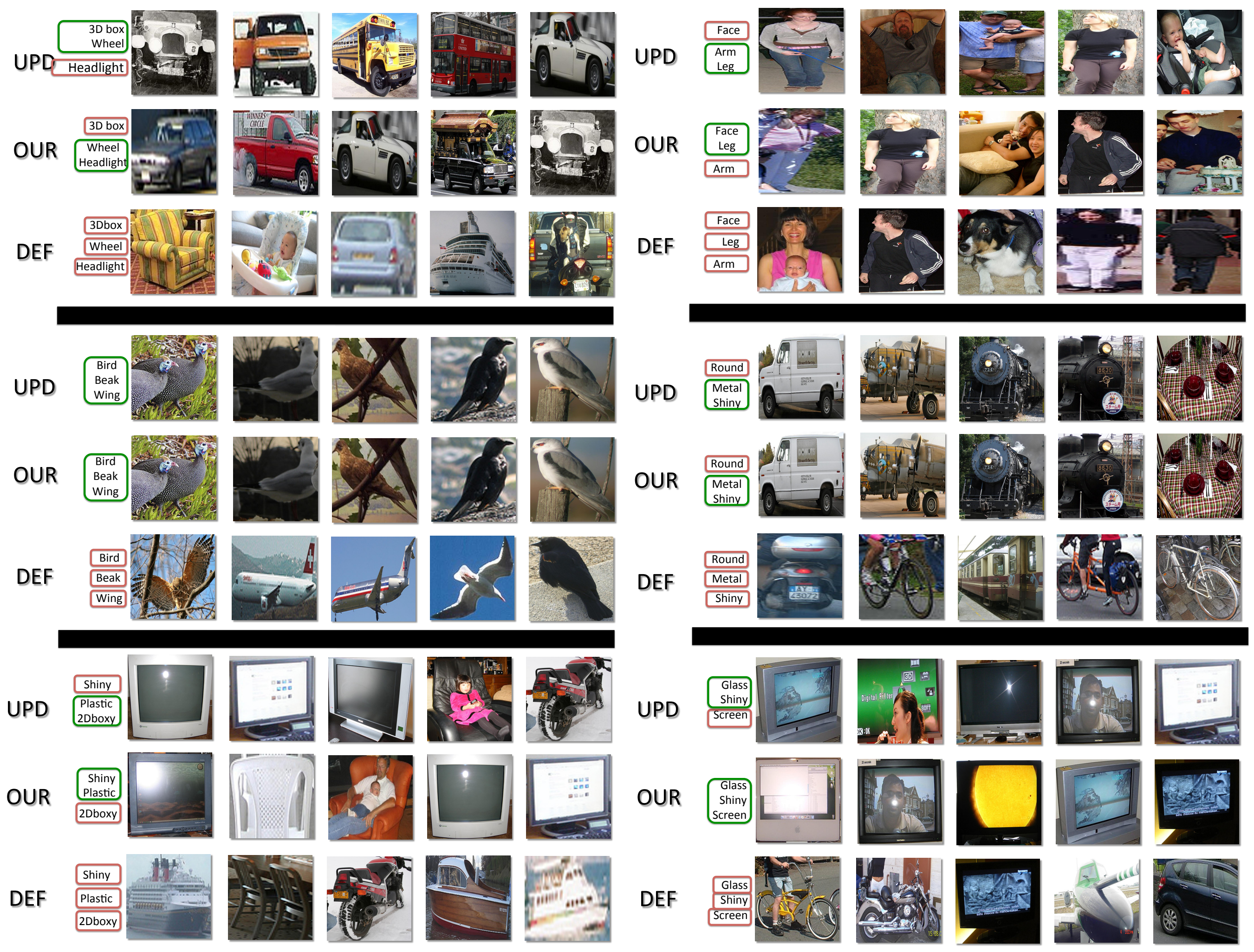


FIGURE 6.10: Qualitative comparisons between our method, the default case and the upper bound. Green boxes correspond to merged classifiers and red ones are for independent classifiers. It is interesting to see that when considered beak, wing and bird independently, retrieved images are mixed between planes and birds. This is due to the labeling in aPascal that both birds and planes wing and beaks are labeled with the same label. Once merged with bird the classifier can find the right images.

Chapter 7

Domain Adaptive Classification Using Predictable Binary Attributes

7.1 Overview

Discriminative learning algorithms rely on the assumption that models are trained and tested on the data drawn from the same marginal probability distribution. In real world applications, however, this assumption is often violated and results in a significant performance drop. For example, in visual recognition systems, training images are obtained under one set of lighting, background, view point and resolution conditions while the recognizer could be applied to images captured under another set of conditions. In speech recognition, acoustic models trained by one speaker need to be used by another. In natural language processing, part-of-speech taggers, parsers, and document classifiers are trained on carefully annotated training sets, but applied to texts from different genres or styles where there is mismatched distributions of words and their usages.

For these reasons domain adaptation techniques have received considerable attention in machine learning applications. Some previous efforts [129–132] consider *semi-supervised* domain adaptation where some labeled data from the target domain is available. We focus on the *unsupervised* scenarios when there is no labeled data from the target domain available. Some earlier work in unsupervised domain adaptation assumes that there are discriminative ”pivot” features that are common to both domains [133, 134]. While such methods might work well in language domains, in visual world typical histogram-based image descriptors (visual words) can change significantly across domains. A recent work [135] considers the labeled source data at the instance level to detect a subset of them (landmarks) that could model the distribution of the data in the target domain well. A drawback of such methods is that they do not use the information

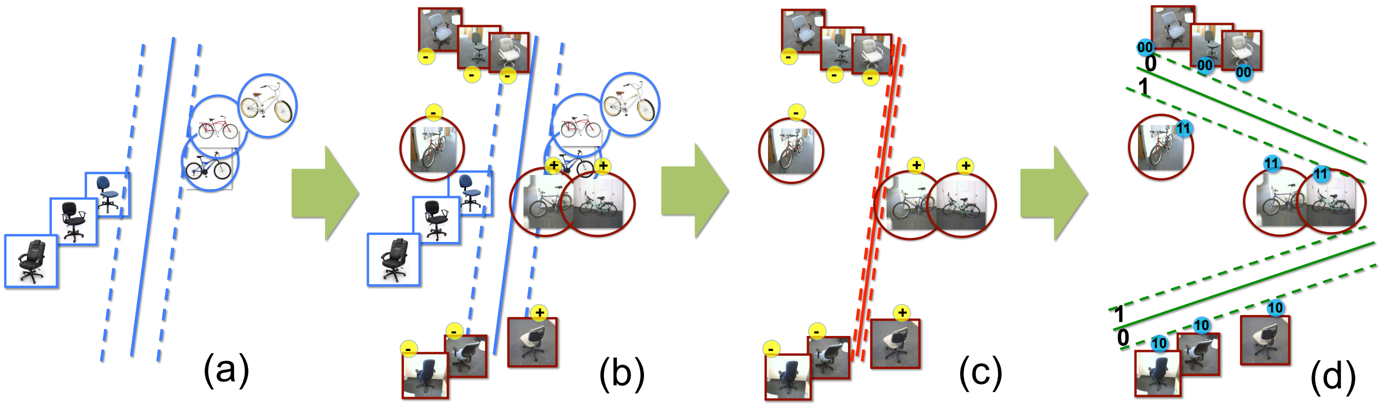


FIGURE 7.1: This figure summarizes the overall idea of our method. (a) shows a classifier that is trained on data for two categories from the **source** domain (internet images). In (b) we classify the data from the **target** domain (webcam images) using the classifier trained in (a). In (c) and (d) we want to use roughly predicted labels in the target domain to find hyperplanes that are discriminative across categories and also have large margins from samples. (c) illustrates a hyperplane that perfectly separates positive and negative samples but has a small margin. (d) shows two hyperplanes that are not perfectly discriminative but they are binarizing data in the target domain with a large margin. The binarized samples by these two hyperplanes are linearly separable.

from all the samples in the source domain available for training the classifier, as they use only landmark points and prune the rest.

Another research theme in domain adaptation is to assume there is an underlying common subspace [136–138] where the source and target domains have the same (or similar) marginal distributions, and the posterior distributions of the labels are also the same across domains. Hence, in this subspace a classifier trained on the labeled data from the source domain would likely perform well on the target domain. However, transforming data only with the goal of modeling the target domain distribution does not necessarily result in accurate classification. Our goal is to identify a transformation that not only models the distribution of a target domain, but also is discriminative across categories.

We propose a simple yet effective adaptation approach that directly learns a new feature space from the unlabeled target data. This feature space is optimized for classification in the target domain. Motivated by [139], our new feature space, composed of binary attributes, is spanned by max-margin non-orthogonal hyperplanes learned directly on the target domain. Our new binary feature sets are discriminative and at the same time are robust against the change of distributions of data points in the original feature space between the source and target domains. We refer to this property as *predictability*. The notion of predictability is based on the idea that subtle variations of the data point positions in the original space should not result in different binary codes. In other words, a particular bit in the binary code should be identical (predictable) for all the data samples that are close to each other in the feature space. Figure 7.1 illustrates the essential idea behind our approach.

Our experimental evaluations show that our method significantly outperforms state-of-the-art results on several benchmark datasets which are extensively studied for domain adaptation. In fact in many cases we even reach the upperbound accuracy that is obtained when the classifier is trained and tested on the target domain itself. We also investigate the dataset bias problem, recently studied in [140, 141]. We show that our adaptive classification technique can successfully overcome the bias differences between the datasets in cross-dataset classification tasks. The joint optimization criteria of our model can be solved efficiently and is very easy to implement. Our MATLAB code is online available[1].

7.2 Background

While it is still not clear how exactly to quantify a domain shift between the train (source) and test (target) data sets, several methods have been devised that show improved performance for cross-domain classification.

In language processing, Daume et al [142] model the data distribution corresponding to source and target domains as a common shared component and a component that is specific to the individual domains. Blitzer et al [133, 134] proposed a structural correspondence learning approach that detects some pivot features that occur frequently and behave similarly in both domains. They used these pivot features to learn an adapted discriminative classifier for the target domain. In visual object recognition, Saenko et al [129] proposed a metric learning approach that uses labeled data in the source and target domains for all or some of the corresponding categories to learn a regularized transformation for mapping between the two domains.

In unsupervised settings where there is no label information available from the target domain, several methods have been recently proposed. Pan et al [136] devise a dimensionality reduction technique that learns an underlying subspace where the difference between the data distributions of the two domains is reduced. However they obtain this subspace by aligning distribution properties that are not class-aware; therefore it does not guarantee that the same class from separate domains will project onto the same coordinates in the shared subspace. Gopalan et al [137] take an incremental learning approach, following a geodesic path between the two domains modeled as points on a Grassmann manifold. Gong et al [138] advance this idea by considering a kernel-based approach; i.e. they integrate an infinite number of subspaces on that geodesic path rather than sampling a finite number of them. In [135], Gong et al, however, consider only a subset of training data in the source domain for their geodesic flow kernel approach; the ones that are distributed similarly to the target domain, .

[1]http://www.umiacs.umd.edu/ mrastega/paper/dom.zip

In [140, 141], the varying data distribution between the train and test sets have been studied under the ”dataset bias” They point out how existence of various types of bias, such as capture and negative set bias, between datasets can hurt visual object categorization. This is a similar problem to domain adaptation where each dataset can be considered as a domain.

Another set of related methods are those that use binary code descriptors for recognition. Farhadi et al [143, 144] used binary feature for supervised transfer learning. Recent method shows that even with a few bits of binary descriptor one can reach state-of-the-art performance in object recognition. Gong et al [51] optimized to find a rotation of data that minimizes binary quantization error. They used CCA in order to leverage labels' information. In [139] they proposed a technique to map the data into a hamming space where each bit is predictable from neighboring visual data. At the same time the binary code of an image needs to be discriminative across the categories. Our method is motivated by their approach. We are also looking for a set of discriminative binary codes but in our problem data comes from different domains with mismatched distributions in the feature space. In section 7.3 we explain how our method solves this problem by a joint optimization over solving a linear SVM and finding a binary projection matrix.

7.3 Adaptive Classification

Our goal is to identify useful information for classification in the target domain. We represent this information by a number of hyperplanes in the feature space created using data from the target domain. We call each of these hyperplanes, an *attribute*. These attributes must be discriminative across categories and predictable across domains. We explain our notion of predictability in section 7.3.2. We use these attributes as feature descriptors and train a classifier on the labeled data in the source domain. When we apply this classifier to the target domain, we achieve a much higher accuracy rate than the baseline classifier for the target data. The baseline is simply a classifier trained on the source data in the original feature space.

Each attribute is a hyperplane in feature space; it divides the space into two subspaces. We assign a binary value to each instance by its ”sidedness” with respect to the hyperplane. We construct a K-bit binary code for each image using K hyperplanes. To produce consistent binary codes across domains, each binary value needs to be predictable from instances across domains. Predictability is the key to the performance of our method. We also want the attributes to be discriminative across categories. i.e. the K-bit attribute descriptors of the samples from same category should be similar to each other and different from the other categories.

7.3.1 Problem Description

First we explain the notations that we use throughout this section. Superscripts $\mathcal{S}$ and $\mathcal{T}$ indicates source and target domains respectively and superscript T indicates matrix transpose. x_i is a d-dimensional column vector that represents the i^{th} instance feature and X is a matrix created by concatenation of all x_i's. l_i is the category label of the i^{th} instance. Without loss of generality, we assume that $l_i \in \{1, -1\}$. A is a $d \times K$ matrix whose k^{th} column, a_k, is the normal vector of a hyperplane (attribute) in the original feature space. w is the K-dimensional normal vector of a classifier that classifies one category from the others in the binary attribute space. sign(.) is the sign function

We want to directly optimize for better classification in the target domain. Therefore, we need to find K hyperplanes, a_k, in the target domain such that when we use sign$(A^T x_i)$ as a new feature space, and learn a classifier on source data projected onto this space, we can predict the class labels of the data in the target domain. Of course we do not have the class labels for the data in the target domain $l_i^{\mathcal{T}}$. In order to train the classifier and attributes (hyperplanes) in target domain, we add a constraint to our optimization to force the $l_i^{\mathcal{T}}$ to be predictable from the source domain's classifier. More specifically, our optimization is a combination of two max-margin SVM-like classifiers that are interconnected via the attribute mapping matrix A.

$$\begin{aligned}
&\min_{A, w^{\mathcal{S}}, w^{\mathcal{T}}, l^{\mathcal{T}}, \xi^{\mathcal{S}}, \xi^{\mathcal{T}}} \|w^{\mathcal{S}}\| + \|w^{\mathcal{T}}\| + C_1 \sum \xi^{\mathcal{S}} + C_2 \sum \xi^{\mathcal{T}} \\
&s.t. \\
&l_i^{\mathcal{S}}({w^{\mathcal{S}}}^T \text{sign}(A^T x_i^{\mathcal{S}})) > 1 - \xi_i^{\mathcal{S}}, \\
&l_j^{\mathcal{T}}({w^{\mathcal{T}}}^T \text{sign}(A^T x_j^{\mathcal{T}})) > 1 - \xi_j^{\mathcal{T}}, \\
&l_j^{\mathcal{T}} = \text{sign}({w^{\mathcal{S}}}^T \text{sign}(A^T x_j^{\mathcal{T}})),
\end{aligned} \tag{7.1}$$

It is not straightforward to solve the optimization in Eq 7.1 because matrix A in the constraints requires a combinatorial search for the optimal solution. But if we constrain the possible solutions for A, then we can solve it efficiently. As we will explain in section 7.3.2, we do this by forcing predictability constraints on all the a_k vectors.

7.3.2 Predictability

In different domains data appears with different distributions. Consider a picture of a car taken by a mobile phone's camera and the same picture taken from a professional high quality camera. Due to differences in the two photo capturing systems such as resolution, the two images will be mapped to two different points in visual feature space despite being the same object from the same category. For better classification, however, ideally we would like to create a feature space

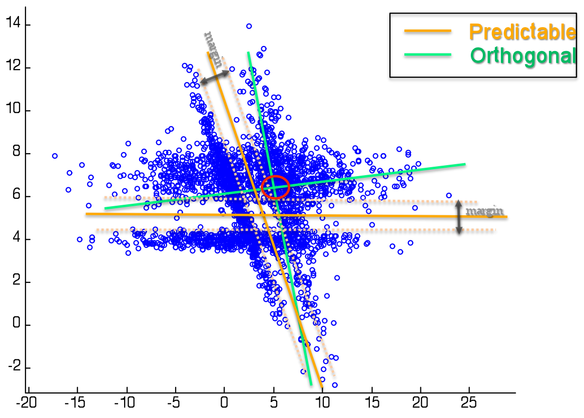


FIGURE 7.2: Comparison of predictable hyperplanes and orthogonal hyperplanes. Note that the hyperplanes learned by large margin divide the space, avoiding the fragmentation of sample distributions by the help of *predictability* constraints implemented by max-margin regularization.

that would map these two images onto the same or nearby points. In other words, we would like to have a class-compact and domain-invariant feature space for these images. For a sample, an attribute is a binary value derived from a hyperplane in the raw feature space. If this hyperplane produces different binary values for samples that are nearby to each other, then we say that the values coming from this hyperplane are not predictable. Therefore, this attribute would not be robust against the variations of samples from different domains in the raw feature space.

Predictability is the ability to predict the value of a given bit of a sample by looking at the corresponding bit of the nearest neighbors of that sample. For example, if the k^{th} bit in most of the nearest neighbors of a sample is **1** then we can infer that the k^{th} bit of that sample would also be **1**.

Consider the situation where a hyperplane crosses a dense area of samples. There would be many samples in proximity to each other that are assigned different binary values. The binary values obtained by this hyperplane are thus not *predictable*. The binary values obtained by a hyperplane are *predictable* when the hyperplane has large margin from samples. There are several methods that try to model the transfer of distribution between domains [136–138]. All of these methods rely on discovering some orthogonal basis of the feature space such as principle components. However these orthogonal basis are not appropriate as hyperplanes for attributes. Figure 7.2 illustrates a demonstration of the hyperplanes defined by orthogonal basis (PCA) in green lines. Note that PCA hyperplanes cross dense areas of samples. If we binarize the samples by the PCA hyperplanes, then samples in the red circle will have different binary codes even though they are nearby each other and strongly clustered. The hyperplanes that are shown in orange are our predictable attributes, which enforce the large margins from samples.

To enforce the predictability constraint on binary values of attributes, we regulate our optimization by adding a max-margin constraint on A as follows:

$$\begin{aligned}
&\min_{A, w^{\mathcal{S}}, w^{\mathcal{T}}, l^{\mathcal{T}}, \xi^{\mathcal{S}}, \xi^{\mathcal{T}}, \xi^{A}} \|w^{\mathcal{S}}\| + \|w^{\mathcal{T}}\| + \|A\|_F + \\
&\qquad C_1 \sum \xi^{\mathcal{S}} + C_2 \sum \xi^{\mathcal{T}} + C_3 \sum \xi^{A} \\
&s.t. \\
&l_i^{\mathcal{S}}({w^{\mathcal{S}}}^T \text{sign}(A^T x_i^{\mathcal{S}})) > 1 - \xi_i^{\mathcal{S}}, \\
&l_j^{\mathcal{T}}({w^{\mathcal{T}}}^T \text{sign}(A^T x_j^{\mathcal{T}})) > 1 - \xi_j^{\mathcal{T}}, \\
&l_j^{\mathcal{T}} = \text{sign}({w^{\mathcal{S}}}^T \text{sign}(A^T x_j^{\mathcal{T}})), \\
&b_{kj} = \text{sign}(a_k^T x_j^{\mathcal{T}}), \\
&b_{kj}(a_k^T x_j^{\mathcal{T}}) > 1 - \xi_{kj}^{A},
\end{aligned} \tag{7.2}$$

Where b_{kj} is the binary value of the k^{th} bit (attribute) of the j^{th} sample in the target domain. In fact, each attribute is a max-margin classifier in feature space and b_{jk} is the label of the j^{th} sample when classified by the k^{th} attribute classifier. This optimization can be easily conducted using block coordinate descent. If we fix $w^{\mathcal{T}}$ and A, then solving the optimization for $w^{\mathcal{S}}$ is a simple linear SVM in the attribute space. Accordingly, once we determine $w^{\mathcal{S}}$, we can compute $l^{\mathcal{T}}$. Then solving for $w^{\mathcal{T}}$ and A is a standard attribute discovery problem in the target domain and can be solved using the method (DBC) in [139]. We iterate over these two steps: finding $w^{\mathcal{S}}$, and then solving for $w^{\mathcal{T}}$ and A. We don't know how to obtain a good initialization for $w^{\mathcal{T}}$ and A, but luckily we don't necessarily need them. We only need to have an initialization for $l^{\mathcal{T}}$ so that we can solve the attribute discovery problem for A and $w^{\mathcal{T}}$. An intuitive way to initialize $l^{\mathcal{T}}$ is to learn a classifier on the labeled data in the source domain, $x^{\mathcal{S}}$ and $l^{\mathcal{S}}$, and then apply it on $x^{\mathcal{T}}$, the data in the target domain. Algorithm 1 summarizes our method.

Algorithm 5 Adaptive Classification

```
Input: $X^{\mathcal{S}}$, $l^{\mathcal{S}}$, $X^{\mathcal{T}}$, $K$.
Output: $l^{\mathcal{T}}$, $A$, $w^{\mathcal{S}}$, $w^{\mathcal{T}}$.
1: $\theta \leftarrow$ Learn a classifier on $X^{\mathcal{S}}$ and $l^{\mathcal{S}}$
2: $l^{\mathcal{T}} \leftarrow$ Test the classifier $\theta$ on $X^{\mathcal{T}}$ //Initialization for $l^{\mathcal{T}}$
3: repeat
4:     $w^{\mathcal{T}}, A \leftarrow$ DBC($X^{\mathcal{T}}$,$l^{\mathcal{T}}$,$K$)
5:     $w^{\mathcal{S}} \leftarrow$ Learn a linear SVM on sign($A^T X^{\mathcal{S}}$) and $l^{\mathcal{S}}$
6:     $l^{\mathcal{T}} \leftarrow$ sign(${w^{\mathcal{S}}}^T$sign($A^T X^{\mathcal{T}}$))
7: until convergence on $l^{\mathcal{T}}$
```

7.4 Experiments

We first evaluate our method on two benchmark datasets extensively used for domain adaptation in the contexts of object recognition [129, 135, 137, 138, 145] and sentiment analysis [134, 135, 137]. We compare our method to several previously published domain adaptation methods. Empirical results show that our method not only outperforms all prior techniques in almost all cases, but also in many cases we achieve the same-domain classification, the upper bound, accuracy, i.e. when the classifier is trained and tested on the target domain itself.

Furthermore, we test the performance of our method on an inductive setting of unsupervised domain adaptation. In the inductive setting we test our adapted classifier on a set of unseen and unlabeled instances from target domain- separate from the target domain data used to learn the attribute model. And finally, we investigate the dataset bias problem, recently studied in [140, 141], and we show that our adaptive classification technique can successfully overcome the bias differences in both single and multiple source domains scenarios.

7.4.1 Cross-Domain Object Recognition

First, we evaluate our method for cross-domain object recognition. We followed the setup of [135, 138] which use the three datasets of object images studied in [129, 137, 145]: Amazon (***A***) (images downloaded from online merchants), Webcam (***W***) (low-resolution images taken by a web camera), and DSLR (***D***) (high-resolution images taken by a digital SLR camera) plus Caltech-256 (***C***) [146]as a fourth dataset. Each dataset is regarded as a domain. The domain shift is caused by factors including change in resolution, pose, lighting, background, etc. The experiments are conducted on 10 object classes common to all 4 datasets. There are 2533 images in total and the number of images per class ranges from 15 (in DSLR) to 30 (Webcam), and up to 100 (Caltech and Amazon). We used the publicly available feature sets [2], and the same protocol as in all the previous work were used for representing images: The 64-dimensional SURF features [147] were extracted from the images, and a codebook of size 800 was generated by k-means clustering on a random subset of Amazon database. Then, the images from all domains are represented by an 800-bin normalized histograms corresponding to the codebook.

We report the results of our evaluation on all 12 pairs of source and target domains and compare it with methods as reported in [135] (Table 1). The other methods include transfer component analysis (tca) [136], geodesic flow sampling (gfs) [137], Geodesic Flow Kernel (gfk)[138], structural correspondence learning (scl)[133], kernel mean matching (kmm) [148], and a metric learning method (metric) [129] for semi-supervised domain adaptation, where label information (1 instance per category) from the target domains is used. We also report a baseline results of no adaptation, where we train a kernel SVM on labeled data from the source domain in the original feature space. A linear kernel function is used for the SVM. For each pair of domains the performance is measured by classification accuracy (number of correctly classified instances over total test data from target).

As explained in [135], due to its small number of samples (157 for all 10 categories), DSLR was not used as a source domain and so the results for other methods have been reported only

[2]http://www-scf.usc.edu/ boqinggo/da.html

	$A \to C$	$A \to D$	$A \to W$	$C \to A$	$C \to D$	$C \to W$	$W \to A$	$W \to C$	$W \to D$	$D \to W$	$D \to C$	$D \to A$
No Adaptation	41.7	41.4	34.2	51.8	54.1	46.8	31.1	31.5	70.7	38.2	34.6	38.2
TCA [136]	35.0	36.3	27.8	41.4	45.2	32.5	24.2	22.5	80.2	N/A	N/A	N/A
GFS [137]	39.2	36.3	33.6	43.6	40.8	36.3	33.5	30.9	75.7	N/A	N/A	N/A
GFK [138]	42.2	42.7	40.7	44.5	43.3	44.7	31.8	30.8	75.6	N/A	N/A	N/A
SCL [133]	42.3	36.9	34.9	49.3	42.0	39.3	34.7	32.5	83.4	N/A	N/A	N/A
KMM [148]	42.2	42.7	42.4	48.3	53.5	45.8	31.9	29.0	72.0	N/A	N/A	N/A
Metric [129]	42.4	42.9	49.8	46.6	47.6	42.8	38.6	33.0	**87.1**	N/A	N/A	N/A
Landmark [135]	45.5	47.1	46.1	56.7	**57.3**	**49.5**	40.2	35.4	75.2	N/A	N/A	N/A
Ours	**47.2**	**48.7**	**46.3**	**57.65**	49.68	44.1	**40.4**	**35.5**	84.0	**86.1**	**35.5**	**39.7**

TABLE 7.1: **Cross-domain Object recognition:** accuracies for all 12 pairs of source and target domains are reported (C: Caltech, A: Amazon, W: Webcam, and D: DSLR). Due to its small number of samples, DSLR was not used as a source domain by the other methods and so their results have been reported only for 9 pairings. Our method significantly outperforms all the previous methods except for 2 out of 3 cases when DSLR , whose number of samples are insufficient for training our attribute model, is the target domain.

for 9 out of 12 pairings. Table 1 shows that our method outperforms all the previous methods in all cases except when DSLR is the target domain. The culprit is the small number of samples in DSLR being insufficient for training the attribute model. In all our experiments, we used a binary attribute space with 256 dimensions. To learn each attribute hyperplane we used linear SVM coupled with kernel mapping. None of the hyperparameters for SVM classifiers and DBC model were tuned. They were all left at their default values. One might get better results by tuning these parameters.

7.4.2 Cross-Domain Sentiment Analysis

Next, we consider the task of cross-domain sentiment analysis in text [134]. Again we compare the performance of our approach with the same set of domain adaptation methods as reported in [135] and listed in section 7.4.1. We used the dataset in [134] which includes product reviews from amazon.com for four different products: books (***B***), DVD (***D***), electronics (***E***), and kitchen appliances (***K***). Each product is considered as a domain. Each review has a rating from 0 to 5, a reviewer name and location, review text, among others. Reviews with rating higher than 3 were classified as positive, and those less than 3 were classified negative. The goal is to determine whether the process of learning positive/ negative reviews from one domain, is applicable to another domain. We used the publicly available feature sets for the collection in which bag-of-words features are used and the dimensionality of data is reduced to 400 (the 400 words with the largest mutual information with the labels).

Table 7.2 shows the results; our method outperforms all the previous methods by a relatively large margin (25% average improvement over baseline and 19% over state-of-art).

	$K \rightarrow D$	$D \rightarrow B$	$B \rightarrow E$	$E \rightarrow K$
No Adaptation	72.7	73.4	73	81.4
TCA [136]	60.4	61.4	61.3	68.7
GFS [137]	67.9	68.6	66.9	75.1
GFK [138]	69.0	71.3	68.4	78.2
SCL [133]	72.8	76.2	75.0	82.9
KMM [148]	72.2	78.6	76.9	83.5
Metric [129]	70.6	72.0	72.2	77.1
Landmark [135]	75.1	79.0	78.5	83.4
Ours	**92.1**	**93.15**	**94.94**	**95.65**

TABLE 7.2: **Cross-Domain Sentiment Classification:** accuracies for 4 pairs of source and target domains are reported. K: kitchen, D: dvd, B: books, E: electronics. Our method outperforms all the previous methods.

7.4.3 Comparing to Same-Domain Classification

How accurate are the domain adapted classifiers compared to classifiers trained on labeled data from the target domain? To investigate this, we divide each dataset into two equal parts, one of which is used for training and the other for testing. This balances the number of samples used for within domain training and testing and cross domain adaptive training and testing.

Table 7.3 shows the results for all 16 pairs of domains in sentiment dataset and 4 pairs of domains from object recognition datasets. In the latter we could use only the two domains (Caltech, Amazon) that had sufficient number of samples to be divided into two groups (train/test)

The rows correspond to the source domains and columns to the target domains. We can see how on this data set our adaptive classification method reaches the upper bound performance in all cases.

7.4.4 Transductive vs Inductive Cross-Domain Classification

In the previous experiments, we follow the same protocol as [135, 138] for a fair comparison. So, we had access to all the samples in the target domain at training time and our goal was to predict their labels. This is a transductive learning problem except that the test data was drawn from a different domain. In an inductive setting we do not have access to the test data at training time. So, to create an inductive setting for the unsupervised domain adaptation problem, we make only a fraction of the data from the target domain accessible at training time for learning our adaptive feature space. The rest, which we refer to as out-of-sample data from the target domain, is set aside for inductive classification tests.

Table 7.4, reports the results for this experiment on the sentiment data set where we have balanced number of samples across domains. Our adaptive classification results on out-of-sample data still outperform the corresponding performance for in-sample data by other methods in 3

	K	E	B	D
K	97.9	97.4	96.6	95.2
E	97.9	97.4	96.5	95.4
B	97.8	97.4	96.6	95.3
D	97.7	97.3	96.6	95.4

	C	A
C	75.6	92.2
A	74.4	92.2

TABLE 7.3: **Comparing to Same-Domain Classification :** (Left) Accuracies for all 16 pairs of source and target domains in sentiment dataset are reported in the left table. K: kitchen, D: dvd, B: books, E: electronics. (Right) Accuracies for 4 pairs of source and target domains are reported. C: Caltech, A: Amazon. Rows and columns correspond to source and target domains respectively. Our method reaches the upper bound accuracies (diagonal) for cross-domain classification.

		$K \to D$	$D \to B$	$B \to E$	$E \to K$
In-samples	No Adaptation	72.7	77.1	75.2	82.8
	Adapted (Ours)	**97.2**	**96.6**	**98.0**	**98.1**
Out-samples	No Adaptation	70.5	75.6	74.4	82.8
	Adapted (Ours)	**77.5**	**76.9**	**80.7**	**84.4**

TABLE 7.4: **Transductive vs Inductive Cross-domain Classification:** The first two rows show the results in transductive setting where all the data from the target domains are accessible during training. The last two rows show the results in inductive setting where we test our classifier only on a subset of data in the target domain that was not accessible during training time

out of 4 cases. Nevertheless, it does show a drop in performance compared with our own in-sample results. As we show later, however, this is not necessarily the case. In section 7.4.5 we show how our out-of-sample results reasonably perform compared to the corresponding in-sample ones. (table 7.5)

7.4.5 Dataset Bias

Most of the images in the datasets studied in sections 7.4.1 and 7.4.2 contain the object of interest centered and cropped on a mostly uniform background. To evaluate our method on a wider range of images with unconstrained backgrounds and clutter, as well as to see how it deals with the data set bias problem addressed in [140, 141], we extend our cross-domain object recognition experiments to four widely used computer vision datasets- Pascal2007 [149], SUN09 [150], LabelMe [151], Caltech101 [146].

We follow the same protocol as [141], where they run experiments on five common object categories- ”bird”, ”car”, ”chair”, ”dog”, and ”person”. We used the publicly available feature sets for this data [3]. Using a bag-of-words representation, Grayscale SIFT descriptors [152] at multiple patch sizes of 8, 12, 16, 24 and 30 with a grid spacing of 4 were extracted. Using k-means clustering on randomly sampled descriptors from the training set of all datasets, a codebook of size 256 is constructed. The baseline SVM is implemented using Liblinear [92]

[3]http://undoingbias.csail.mit.edu/features.tar

		Caltech	LabelMe	Pascal07	SUN09
In-samples	No Adaptation	78.7	71.6	76.1	70.9
	Adapted (Ours)	**99.4**	**92.7**	**92.6**	**94.9**
Out-samples	No Adaptation	79.1	75.1	75.0	74.2
	Adapted (Ours)	**94.6**	**86.4**	**90.1**	**87.8**

TABLE 7.5: **Cross-Dataset Object Recognition:** The 4 rightmost columns show the classification results for when we hold out one dataset as the target domain and use the other 3 as source domains, in both the inductive (first two rows) and transductive (last two rows) settings. The reported results are averaged over 5 categories of objects.

coupled with a Gaussian kernel mapping function [153]. The results are evaluated by average precision (AP).

Table7.5 reports the results of our cross-dataset classification in both the inductive (in-sample) and transductive (out-of-sample) settings. Each column of the table correspond to the situation where one dataset is considered as the target domain and all the remaining datasets are considered as the source domain (multi-source domain). These result shows that our approach is robust against varying biases when the training data comes from multiple datasets and the test data comes from another one. The reported results are averaged over all 5 categories. The average performance improvement by our adaptive method over the baseline (no adaptation) is 28% for out-of-sample data and 18% for in-sample data. The only related work that we are aware of that has performed theses cross-dataset classifications experiments with the same settings is [141] where they report an average performance improvement of only 2.5% across all datasets and all categories.

7.4.6 Effectiveness of Predictability

Now, we show the importance of the predictability of attributes by quantitative and qualitative evaluations.

Quantitative evaluation: To see how learning binary attributes by itself is contributing to our performance increase, we ignore the adaptation and use the attribute features learned only from the source domain. In this setting we learn the binary attribute space from the labeled data in the source domain, project the data from both source and target domain onto this space where we train a classifier on the source data and test it on the target data. We then compare the results with corresponding ones by our adapted model. We used the same experiment setup in section 7.4.5 for this evaluation (Figure 7.3).

Qualitative evaluation: Here we show that the discovered attributes are consistent across domains. We pick an attribute classifier learned by our method, then we find images (from both source and target) that are most positively and negatively confident when classified by this attribute classifier. In Figure 7.4 the left two rows use DSLR as source domain and Amazon as

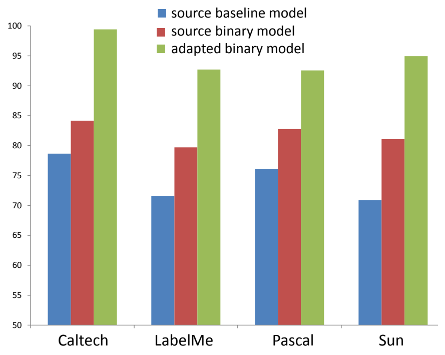


FIGURE 7.3: **Quantitative Evaluation of Predictability:** The blue bars show the classification accuracies when the classifier is simply trained on the data from the source domain in original feature space (baseline). The red bars show the results when the classifier is trained in a binary attribute space learned from the data in the source domain (source binary). The green bars show the results of our adapted model when the classifier is trained on labeled source data in a binary attribute space learned in the target domain (adapted binary). In average the source binary model is increasing the performance by 10% over the baseline while the adapted binary model does that by 28%

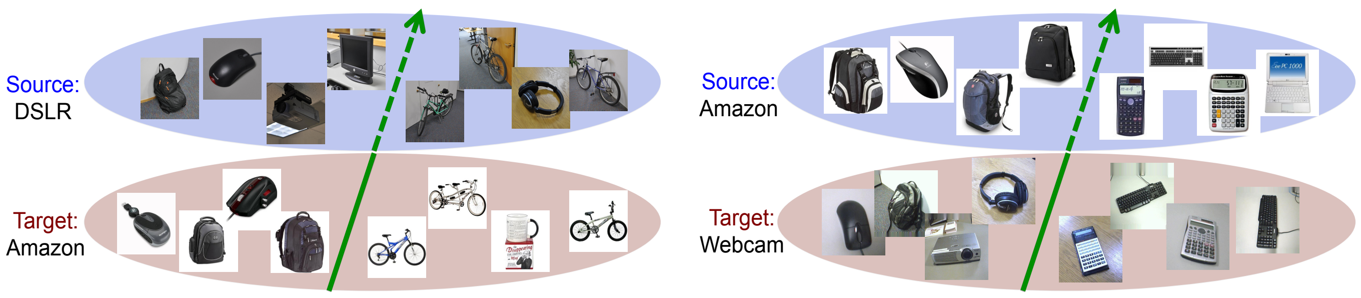


FIGURE 7.4: **Qualitative Evaluation of Predictability:** This figure illustrates two examples where an attribute hyperplane (green arrow), learned by our joint optimization, discriminates visual properties consistently across two different domains. In the left case, the hyperplane is discriminating between the objects with round shapes vs the ones with more surface area. In the right example, the hyperplane is discriminating the keypad-like objects against the more bulky ones. The dashed part of the arrow indicates that the same hyperplane which is trained in target domain is applied in the source domain.

target. Similarly, the right two rows use Amazon as source and Webcam as target. The green arrow represent an attribute classifier which is trained on target domain. The dashed part of the arrow illustrates that the same hyperplane which is trained in target domain is applied in the source domain. Images on the right side of the green arrow are the most positive and on the left side are the most negative one. As can be seen in both cases the attribute classifiers are consistent across domains. In the first case, the attribute consistently separates round shapes from dark-volumed shapes in both domains and in the second case, the attribute consistently discriminates between objects with keypad and objects with dark-volumed shape. This observation is consistent with our intuition of predictability in our optimization.

Chapter 8

Evaluation of Binary Coding Methods

8.1 Overview

Binary codes are attractive representations of data for similarity based search and retrieval purposes, due to their storage and computational efficacy. For example, 250 million images can be represented by 64 bit binary codes by employing only 16 GB of memory. Hashing is a common method to convert high dimensional features to binary codes whose Hamming distances preserve the original feature space distances. Although shorter codes are more desirable due to direct representation in hash tables, longer binary descriptors of data have also been shown to be efficient for fast similarity search tasks. For example, Norouzi et al. [154] proposed a multi-index hashing method, and Rastegari et al. [2] introduced a branch and bound approach to perform exact k-nearest neighbors search in sub-linear time with long binary codes.

There are two major categories of hashing methods. One group is based on random projections of data. For example, Datar et al. [155] introduced locality sensitivity hashing (LSH) and provide theoretical guarantees on retrieval in sub-linear time. Locality Sensitive Hashing refers to a wide range of techniques but we use the term LSH for the random projection based technique as is common in the vision and learning communities. Data driven approaches, on the other hand, employ a learning procedure for binary code mapping. For instance, Weiss et al. [156] introduced Spectral Hashing (SPH) and [157] Multidimensional Spectral Hashing (MDSH); they formulate the problem as an optimization that reduces to an eigenvalue problem. Furthermore, Gong and Lazebnik [158](ITQ) and Norouzi and Fleet [159] (CK-means) proposed methods to minimize the quantization error of mapping the data to a binary hypercube via rotation. These methods are based on vector quantization. Product Quantization [160] is an instance of Cartesian K-means[159] that does not optimize for rotation and Orthogonal K-means is the binary version of CK-means that number of subspaces are equal to the number of bits. There is another class of binary code learning methods that are supervised [3, 161, 162]. In these methods pairs of similar samples are provided for training and metric is learned. We experimentally observed that given a good supervision, MLH[161] performs similar to ITQ. We restrict our attention to unsupervised methods – there is no specification of similar or dissimilar samples.

A common task used for evaluating performance of binary codes is a range searching task – find all samples within distance r of a query point. The performance of different methods in preserving feature space distances in the Hamming space depends on the distribution of data in the original feature space and the way that data is normalized before codes are designed. Recent papers [157, 159, 163–169] showed comparisons without taking these factors into account. In particular, some methods preserve cosine similarity [158], while others preserve Euclidean distance [157, 163]. We should not compare the performance of these methods blindly.

Our main contribution is to show that in many published comparisons, this distinction has not be accounted for sufficiently. When the goal is preserving cosine similarity, it is necessary to normalize the orriginal features by mapping them to the unit hypersphere before learning the binary mapping functions. On the other hand, when the goal is to preserve Euclidean distance, the original feature data must be mapped to a higher dimension by including a bias term in binary mapping functions. Our experiments reveal very different results in the performance of the binary code methods when these constraints are enforced. We show that the state-of-the-art technique OK-means [159] and MDSH [157] most of the times, in first scenario, performs worse than Iterative Quantization (ITQ)[158], which is an older method. LSH [155] most of the time performs better than every other method in the second scenario for long binary codes.

It is not obvious how to enforce the bias term for methods that preserve cosine similarity. Inspired by [3], we propose a geometrical intuition on learning Iterative Quantization (ITQ)[158] augmented by a bias term. This binary embedding leverages correlation using the notion of predictability introduced by Rastegari et al. [3, 6]. A particular bit in a predictable binary code is (ideally) identical in all neighboring data samples. Rastegari et al. [3, 6] use predictability to represent nearest neighbor preservation via a max-margin formulation. However, they had supervision in the form of specification on similar samples which results in tractable optimization. We show that for unsupervised discovery of predictable binary codes the max-margin formulation is intractable. But, by employing a bias term in ITQ we can easily produce predictable codes. Further, we show that even more predictable binary codes can be obtained by random perturbations of the hyperplane parameters at each iteration of quantization. Our experimental evaluation on three datasets shows that augmenting ITQ with a bias term (Predictable Hashing) often outperforms LSH and state-of-the-art data-driven methods.

The principle contribution of this work is methodological – a specification of appropriate methodology for comparing binary coding methods. When these methods are employed, the relative performance of coding algorithms is often very different from results reported in the literature.

Additionally, we propose a variation of iterative quantization (ITQ), using predictability of the bits, which involves extending the ITQ algorithm with a bias term.

8.2 Experimental Settings and Notations

In order to evaluate the accuracy of binary codes we measure precision and recall for the task of r-nearest neighbor retrieval. First we partition the data into two sets; train and test. On the training partition we learn the binary codes and on the testing partition we use that model to extract the binary codes. We take the actual nearest neighbors in the original feature space as ground truth. Then we use the binary codes to find the nearest neighbors in binary space using Hamming distance. Given a query sample, we retrieve the samples around the query in the Hamming space within radius r. For each r we compute the precision and recall. Therefore by varying r we construct the precision-recall curve.

8.2.1 Datasets

We consider three datasets from the computer vision community for image retrieval tasks and object recognition. **ImageNet20K [170]** ImageNet includes 17000 categories of objects. We used the benchmark of ImageNET referred to as ISLVRC2010 which has 1000 categories; for each category we randomly selected 20 samples. In total we have 20K images. This setting is followed by [3]. We used BoW (with 1000 codewords of SIFT features), SPHoG, GIST and LBP as visual features; we concatenated them into a long feature vector and used PCA to reduce the dimensionality to 1000. **1MGist [160]** This dataset contains 1M internet images and their GIST features; it has 960 dimensions. This dataset was created by Jégou et al. [160] and has been used for approximate nearest neighbor search in the vision community. **SUN14K [171]** This dataset has 700 categories of images. Unlike ImageNet, this dataset is for scene recognition. We used the portion of this dataset used for attribute based recognition by Patterson and Hays [172]. It has 14K images and we used the visual features extracted by Patterson and Hays [172] which have 19K dimensions. We randomly selected 1K dimensions.

8.2.2 Methods

We compare different state-of-the-art binary embedding methods on the image retrieval task. The methods we evaluate are: Iterative Quantization (**ITQ**) [158], Orthogonal K-means(**OK-means**)[159], Spectral Hashing (**SPH**) [156], Multidimensional Spectral Hashing (**MDSH**) [157], Locality Sensitive Hashing (LSH) [173]; we compare with two variant of LSH – one with the bias term (**LSHbias**) and the other without (**LSHnobias**). We present ITQ augmented with a bias term as Predictable Hashing (**PH**). We used the online available software for ITQ, MDSH and OK-means provided by their authors.

8.2.3 Notation

We denote the data matrix by $X \in \mathbb{R}^{D \times N}$ where D is the dimensionality of the space and N is the number of data points. A data point is represented by x_i which is the i^{th} column of X. The output binary codes are represented by $B \in \{1, -1\}^{K \times N}$ where K is the number of bits and b_i is the i^{th} column of B.

8.3 Preserving Cosine Similarity

Our goal here is to map points that have high cosine similarity in the original feature space to binary codes that have small Hamming distance. If we regard the binary codes as hash buckets, this is equivalent of bucketing or discretizing the orientation in the original feature space. One way to accomplish this is to use LSH, but force each random hyperplane to pass through the origin (We assume the data are mean centered, i.e. mean of the data is at the origin). In LSH, each data point is projected to a K dimensional subspace using K random projections. Each projection is a hyperplane in the original D dimensional space. We denote w_k as the normal vector for the k^{th} hyperplane. The values of the elements of w_k are randomly drawn from a stable distribution. A datapoint x_i in $\mathbb{R}^D$ is binarized to 1 or -1 by a random hyperplane w_k depending on which side of the hyperplane w_k it lies in. This binarization is simply achieved by taking the sign of $w_k^T x_i$. By using K random hyperplanes, we generate a K dimensional binary code for each data point, creating a K dimensional Hamming space.

To compare such a method to one that preserves Euclidean distance, we must make sure that the comparison is fair. This can be ensured by simply normalizing the data by mapping it to the surface of the unit hypersphere. In this case, Euclidean distance is inversely proportional to cosine similarity, $\|x_i - x_j\| = 2(1 - cos(\theta_{ij}))$ where θ_{ij} is the angle between x_i and x_j. Therefore, all methods that preserve Euclidean distance will preserve cosine similarity as well. Unit-norm normalization is a very popular normalization method for visual features. For the intensity based features it provides some robustness to illumination and for high dimensional histogram feature it captures the pattern of the histograms which somewhat ameliorates the curse of dimensionality.

Gong and Lazebnik [158] did not explicitly show that ITQ preserves cosine similarity. It is simple to see that what ITQ is optimizing for is the best rotation of the unit(binary) hypercube so that the nodes of the hypercube have minimum distance from data points $\min_{R,B} \|RX - B\|$

where R is a rotation matrix . When the data are not normalized, there is a large amount of error caused by different magnitudes of data points compared to the locations of the nodes of the hypercube. But when the data is normalized to the hypersphere the only source of error is the angular difference from data points to the nodes of the hypercube. Therefore, ITQ will achieve its best performance when the data are normalized.

In SPH and MDSH the affinity between data points $A(x_i, x_j) = \exp(-\|x_i - x_j\|^2/2\sigma^2)$ is more correlated with Euclidean distance but normalizing the data makes it more correlated with cosine similarity.

Figure 8.1 shows the precision-recall curves for performing nearest neighbor retrieval on each dataset. In each experiment, we used a specific code length and in the ground truth we fixed the number of nearest neighbors. These are reported in the title of each plot. ”Dball” indicates the average number of nearest neighbors and ”nbits” is the number of bits(code length). Figure 8.2 shows the effect of the number of bits on the accuracy of retrieval. We vary the number of bits (Code length) by 32, 64, 128, 256 and 512. For each code length we measure the average precision by varying the radius size.

In Fig 8.1 and Fig 8.2 we show the results of comparing all methods when the data are appropriately normalized. ITQ consistently performs the best. This is contrary to the results presented in [159] and [157] that claimed that OK-means and MDSH outperform ITQ. In the GIST and ImageNet datasets OK-means and ITQ exhibit comparable results. We examined the distribution of eigenvalues of the data in each dataset and observed in GIST and ImageNet the eigenvalues are more uniformly distributed than the SUN dataset. This indicates that the intrinsic dimensions of feature spaces of GIST and ImageNet (which is BoW) are less correlated than the feature space of SUN (which is a combination of BoW, GIST, SSIM and color histogram).

8.4 Preserving Euclidean Distance

Here we train a model so that points with low Euclidian distance in the original feature space have small Hamming distance in the Hamming space. In this scenario the performance of LSH is significantly better than suggested by results published in conjunction with recently developed data driven approaches. When the length of binary codes are sufficiently large, LSH outperforms these data driven methods. This is mainly because these data driven methods need a mechanism avoid selecting hyperplanes that are redundant - that code the training data very similarly. They do this by enforcing that vectors of encoded bits be uncorrelated across hyperplanes. We will explain why this constraint on correlation is too strong, and limits performance of data-driven methods. Furthermore, we investigate different data distributions, and illustrate and reason about the cases in which LSH performs better. We propose a simple coding scheme based on the notion

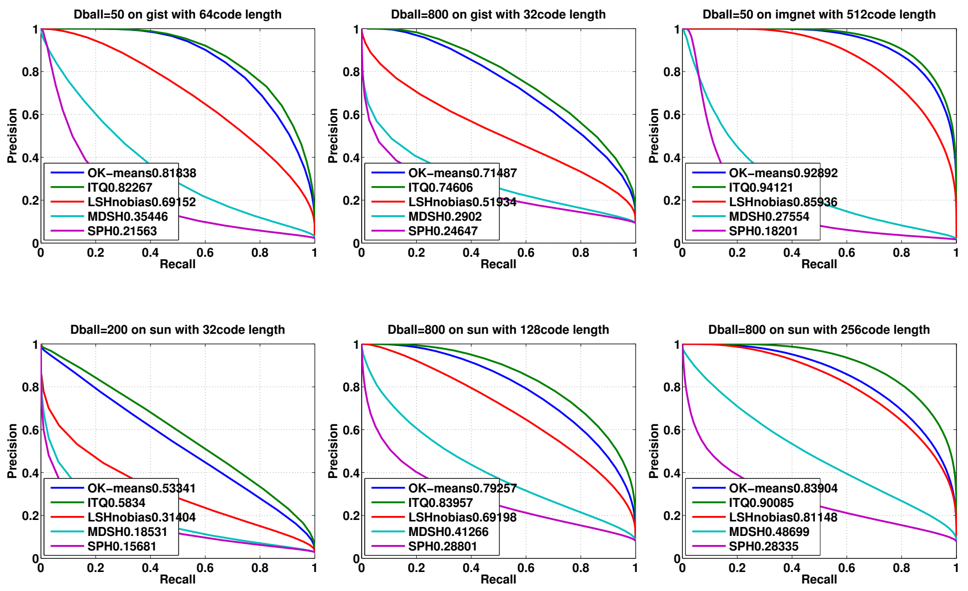


FIGURE 8.1: Precision-Recall curves with different number of NN on different datasets when data are normalized

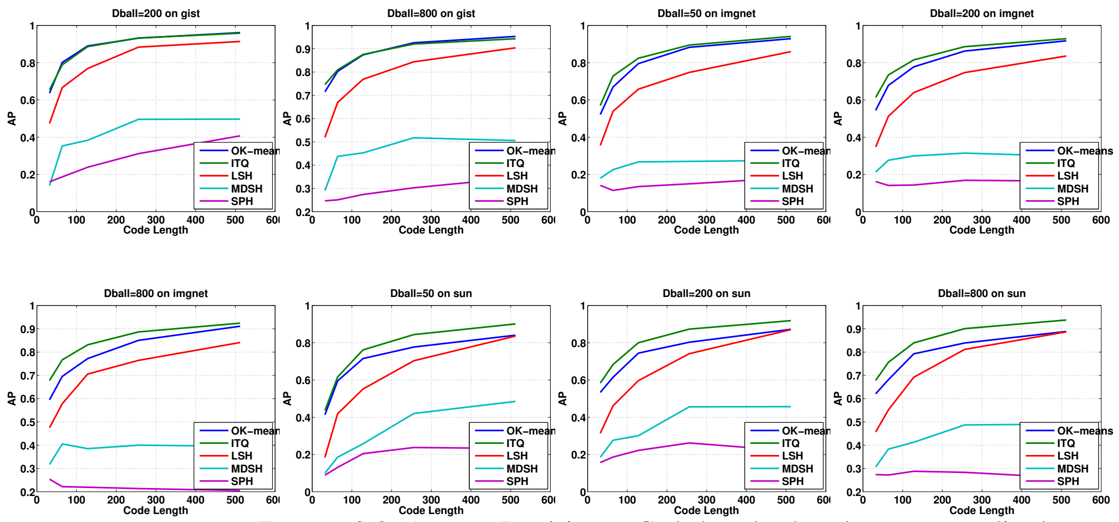
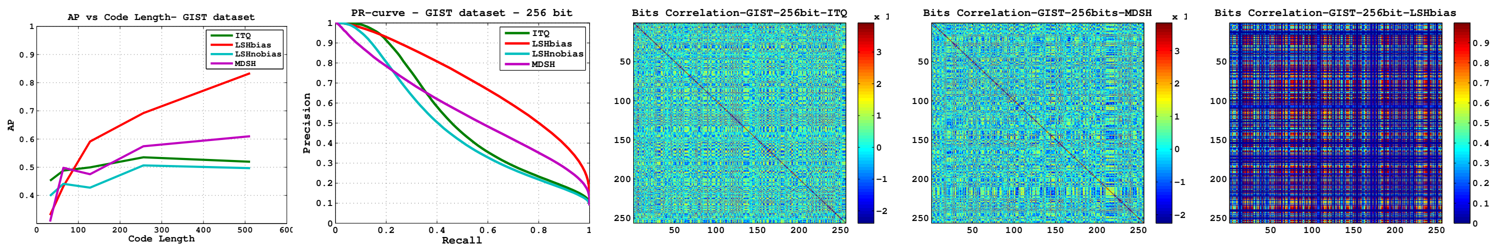


FIGURE 8.2: Average Precision vs Code length when data are normalized

of predictability that allows ITQ to learn binary codes that model the correlation between the original features. We show that predictability allows the method to be data driven without the need to enforce orthogonality of the bits. Our solution outperforms LSH up to fairly large code lengths.

8.4.1 In Defense of LSH

The power of LSH has been underestimated in many recent papers [157, 158, 161]. We assume that nearest neighbors in the original Euclidean space are measured by the ℓ_2-norm. According



(a) (b) (c) (d) (e)

FIGURE 8.3: Impact of the bias term in LSH on GIST dataset: (a) shows average precision vs. code length. (b) shows precision-recall curve at 256bit code length. (c-e) shows the correlation between bits generated by ITQ,MDSH and LSHbias respectively

to Datar et al. [155], the chance of preserving nearest neighbors in the Hamming space by LSH will be higher when the random values of w_k come from a normal distribution.

In LSH each random hyperplane can have a bias term. This bias term can shift the hyperplane away from the origin, but should not move the hyperplane out of the region that the data points lie in. The diameter of the region that contains the data points is defined by d which is the distance between the two farthest points and the radius of the region is $r = \frac{d}{2}$. If we assume that data points are mean centered, then the bias term for each random hyperplane is a real value chosen uniformly from the range $[-r, +r]$. Inspecting the authors' codes in many binary hashing papers for image retrieval [158][1] [174][2] [161][3] [157][4], reveals that they do not include a bias term for random hyperplanes in LSH when comparing their methods with LSH for preserving Euclidean distance.

One may think that since the original feature space for images is usually very high dimensional and the bias term just adds one more dimension, that its impact should be negligible. We show that the impact of the bias term is considerable when the length of the binary code is large. Figure 8.3 (a) shows a comparison on the GIST dataset, which is commonly used in image retrieval. When we increase the number of bits, the performance of LSH with the bias term (LSHbias) significantly increases. LSHbias even outperforms state-of-the-art data driven methods. Two questions arise immediately: *Why do data driven methods perform worse than LSHbias?* and *Why, with short code lengths, does LSHbias not perform as well as with larger code lengths?*

To answer these questions we investigate two well known data driven methods for producing binary codes – Spectral Hashing (SPH) and Iterative Quantization (ITQ). SPH constructs binary codes with uncorrelated bits that minimize the expected Hamming distance between datapoints, i.e. pairs of points that have small distances in the original feature space, should also have small Hamming distances in the binary code space. Their optimization leads to a mapping of data into

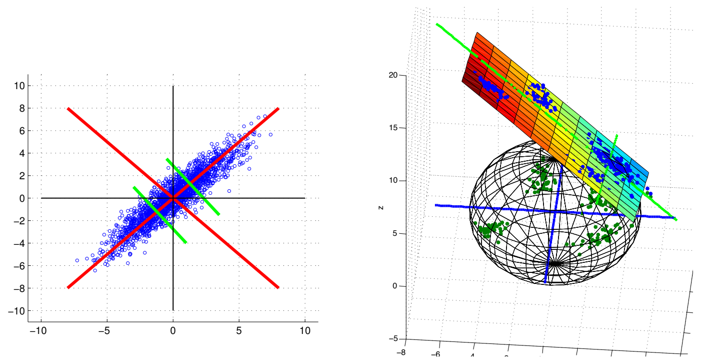
[1] http://www.unc.edu/~yunchao/code/smallcode.zip
[2] http://www.cse.ohio-state.edu/~kulis/bre/bre.tar.gz
[3] https://subversion.assembla.com/svn/min-loss-hashing/trunk
[4] http://www.cs.huji.ac.il/~yweiss/export2.tar

the space of eigen functions. The binary codes are produced by thresholding that mapped data using orthogonal axes in the eigen space. ITQ searches for a rotation of the data (after reducing the dimensionality by PCA) which when theresholded with respect to orthogonal axes results in small quantization error. The orthogonality of the axes leads to uncorrelated bits in the binary codes. Although these two approaches (SPH and ITQ) optimize different objectives, they share the common hard constraint that forces the bits to be *uncorrelated*. This hard constraint actually degrades the power of these methods. Figure 8.3(b) shows the precision-recall curves for 256 bit binary codes of different methods and (c-e) shows the correlation between the bits in ITQ, MDSH and LSHbias respectively in 256 bit codes. The bits in LSHbias have more correlation than MDSH and ITQ, but still LSHbias performs better. This observation is contrary to the popular belief that bits in binary codes should be uncorrelated.

Suppose a bit value for a point in the feature space is created by a hyperplane w which crosses the data points in $\mathbb{R}^D$. In order to generate a K-bit binary code for each sample, we need K hyperplanes. We stack all the normal vectors of the hyperplanes in a $D \times K$ matrix W where the k^{th} column of W is the normal vector for the k^{th} hyperplane. If we want the bits to be uncorrelated then we must have $B^T B \approx I$. The bits are generated by $B = sign(W^T X)$ therefore:

$$B^T B = sign(X^T W) sign(W^T X) \approx I \tag{8.1}$$

By relaxing the binary values (ignoring the sign function), we have $X^T W W^T X \approx I$. Let $S = WW^T$. Now we can rewrite the equation as $X^T S X \approx I$. This indicates that matrix S must be high rank. But since S is generated by the outer product of W with itself, its rank cannot be more than $min(D, K)$ ($W$is $D \times K$). In most binary hashing methods we want the number of bits to be fewer than the original dimensionality of the feature space. Thus $D > K$. Therefore, the rank of S must be K. This means that the K hyperplanes have to be orthogonal. Furthermore, to obtain uncorrelated bits, all the hyperplanes must pass through the mean of the data points. This forces the hyperplanes to have almost equal portions of the data points on each side. Figure 8.4-(a) shows a case of distribution of data in which data points (shown in blue) are highly correlated. To create a 2-bit binary code for this data we use two lines (hyperplanes in $\mathbb{R}^2$). The red lines indicate the orthogonal directions of PCA on this data. By rotating these red lines we can produce all the possible orthogonal directions that cross the mean of the data (In this figure the data are mean centered). However, the best way of splitting these data by two lines (to produce binary codes that preserve the nearest neighbors of each point) is given by the green lines. The green lines are not orthogonal and they do not cross the mean of the data points. Moreover, the bits produced by the green lines are also highly correlated. Here it makes sense that the bits be correlated, because the data itself are highly correlated. In most real world situations there are correlations between data. Therefore, binary codes should indeed model these correlations. The translations of the hyperplanes are very important for modeling



(a) (b)

FIGURE 8.4: (a) on highly correlated data orthogonal splits cannot produce effective binary codes. (b) We search for predictable splits by rotating the augmented plane on the unit sphere. The blue points are the original points in 2D that are mapped to a higher dimension and the green points are the projections of the original points after a proper rotation has been found

these correlations. The bias term in LSH is the key factor, because it allows the hyperplanes to move freely in space.

One way to measure linear correlation between data points is to look at the eigenvalues of the principle components of the data. If all the components have equal eigenvalues, the data are not linearly correlated. There may be nonlinear correlations between the data but capturing nonlinear correlations is not trivial, and, in fact, is an area of current research; one can use Kernel-PCA[175]. If there were no strong correlations between data, then the bias term is not needed. In such a situation, data driven methods with the constraint of uncorrelated bits would work perfectly.

We next investigate the second question; *Why, with short code lengths, does LSHbias not perform as well as with large code lengths?* Consider two data points binarized by LSH into K bits. If all K bits of these two instances are equal, then the probability that these two points are neighbors is $1 - 2^{-K}$. This is simply the confidence of being neighbors given the K bit Hamming similarity of random projection binary codes. This confidence value increases exponentially with K. When K is very small, ITQ performs significantly better than other methods because it first reduces the dimensionality of data to K via PCA. We know that the first few components of PCA usually have similar eigenvalues. This means that when the dimensionality of data is drastically reduced, they will not be highly correlated in the reduced space. For example, consider a 3D space where the data points are uniformly distributed on a plane. When you project the points on the first two eigenvector , which is the plane that fits data the best, the projected points are no longer correlated in the 2D space. Therefore, it is safe to enforce the constraint of uncorrelated bits during code construction.

Now the question is: how can we take correlation into account for learning the binary codes? In the next section we propose a method that uses the notion of predictability and employs a bias term in the hyperplanes for learning in ITQ. We do not claim that our method outperforms LSHbias for all large numbers of bits, but our method shows superior empirical performance using fairly large code length (256 and 512 bits).

8.4.2 Predictable Embedding: Enforcing bias term in ITQ

The concept of predictability in binary codes was introduced for attribute discovery [3] and dual-view hashing [6]. It is the ability to predict the bit value of an arbitrary bit in a binary code of a sample by looking at the same bit of the binary codes of the nearest neighbors of that sample. For example, if the k^{th} bit of a predictable binary codes is 1 then the k^{th} bit of predictable binary codes of the neighbors of that sample are also highly likely to be 1 (in other words, the bit value can be easily predicted as 1). Rastegari et al. [3, 6] proposed that a bit value produced by a hyperplane is predictable when the corresponding hyperplane has max-margin from data points. However, in both their approaches, they need labels for data to learn the max-margin hyperplanes. In [3], their goal was to produce binary codes that preserve the classification performance compared to the original feature space, therefore they use the class labels of each sample. In [6], their goal was to learn a shared Hamming space between two modalities (e.g. textual and visual). They use the bit values in one modality as labels for samples in the other modality. For our purpose, however, unlike the general predictable binary code setting, we do not have label information for samples to learn max-margin hyperplanes. We formulate an optimization function based on a max-margin criteria that does not require label information, as follows:

$$\begin{aligned} &\min_{w,\xi} \|w\|_2^2 + C\sum \xi_i \\ &s.t. \\ &\quad \forall i, \quad sign(w^T x_i)(w^T x_i) > 1 - \xi_i \\ &\quad \forall i, \quad \xi_i > 0. \end{aligned} \tag{8.2}$$

Given that each bit in the space is generated by a hyperplane, we seek a hyperplane w that has max margin from data points X. To enforce the max-margin constraint in equation 8.2 we use $sign(w^T x_i)$ as the label of the i^{th} sample. We embedded the bias term into the x_is by $[x_i; 1]$. Equation 8.2 is combinatorial due to the sign function and so it is not practical to solve this optimization directly.

Predictability involves finding a hyperplane that passes through a region of data points with large margin. ITQ similarly looks for gaps between the main axes (e.g. X and Y which are orthogonal to each other) and the data points. If we can find a mapping of data that has a gap between the mapped points and the main axes, a simple quantization algorithm (ITQ) would construct predictable hyperplanes. To achieve this, we lift all the data points into a space with one more dimension by representing them using homogeneous coordinates. This is nothing more than enforcing a bias term for hyperplanes. To learn these bias terms we simply apply ITQ to the lifted data. Next we present a geometrical interpretation.

For illustration purposes, suppose the data lies on a 2D plane; by adding one more dimension we simply lift up the plane on to $Z = 1$ in the X-Y-Z coordinate system. By rotating this plane with respect to the origin we obtain a configuration in which the gaps between the data points are aligned with the axes in the X-Y coordinate system. Once we achieve that, we project the data back to the X-Y plane. The translation is implicitly determined by rotation in the higher dimension.

Figure 8.4 (b) illustrates the process of adding one more dimension, rotation and projection. As can be seen, the axes on the original planes (green axes) are centered at the mean of the data points (blue points) which is in the dense area of the points. Quantizing the data using these axes will assign different binary codes to the points around the center. By rotating the data on the plane we can never obtain a good quantization from the axes of the plane because the center point is an invariant of such rotations. Thus, the data around the center are always assigned different binary codes (i.e., not predictable) while they are highly clustered. By rotating the plane of lifted data around the origin of the unit sphere and projecting it back to the X-Y plane we obtain the green points which are well separated by the main axes (blue axes). Basically, by rotating in higher dimension we achieve an affine transformation of data in the original space (In this figure X-Y) that can squeeze or stretch the data to align gaps between the data with the main axes.

Formally, we first reduce the dimensionality of the feature vectors to K by PCA and append **1** to X to lift the points into the space that is larger than the original one by 1 dimension. We denote the augmented and dimension-reduced vector by $\hat{X} = [X; \mathbf{1}^T]$ where **1** is a vector of ones. The optimization is $\min_{B,R} \|B - R\hat{X}\|_F^2$ s.t. $RR^T = I$. This optimization is identical to the one in ITQ. We can solve it with same iterative procedure as ITQ. One might think the same strategy would have a similar effect on Orthogonal K-means but it does not. In OK-means, in contrast to ITQ, the optimization for finding the best rotation is in the original feature space, not in the PCA reduced space. In OK-means the space will be divided to K partitions. After finding a global rotation, each partition will be quantized into two parts using a k-means like procedure. Therefore, the bias term will be effective on one partition at a time which is equivalent to being applied to a single bit at each iteration. We have not observed any difference in performance by enforcing a bias term in OK-means; therefore, we do not include them in our results.

Figure 8.5 shows the precision-recall curves for performing nearest neighbor retrieval on each dataset. These results shows the LSH with bias (LSHbias) and ITQ with bias (PH) are outperforming other methods most of the times. Again, this is contrary to results published in previous papers [157, 159] on these methods. It can be observed that OK-means gives better performance only for short binary codes in the GIST and ImageNet datasets, but as we increase the number

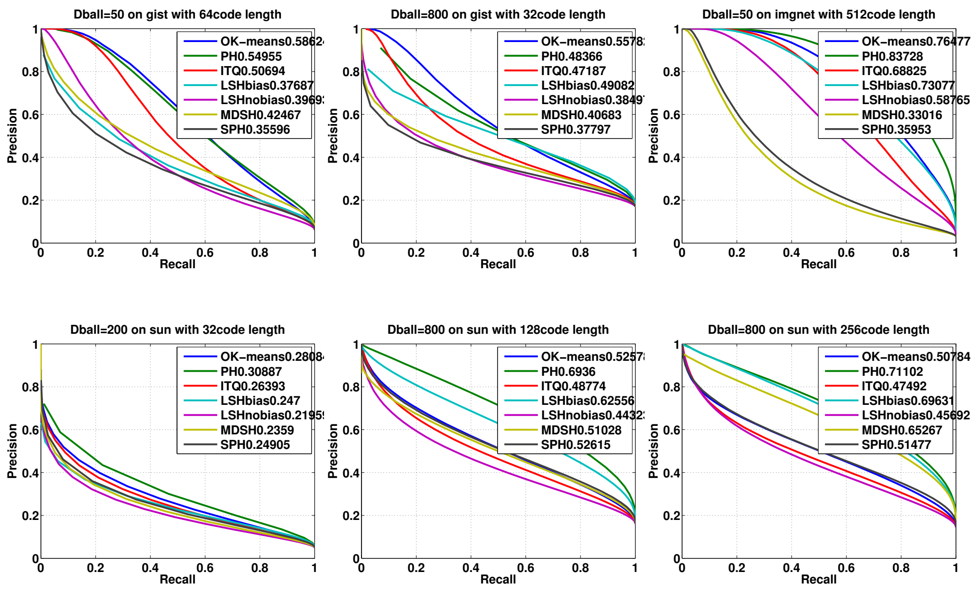


FIGURE 8.5: Precision-Recall curves with different number of NN on different datasets

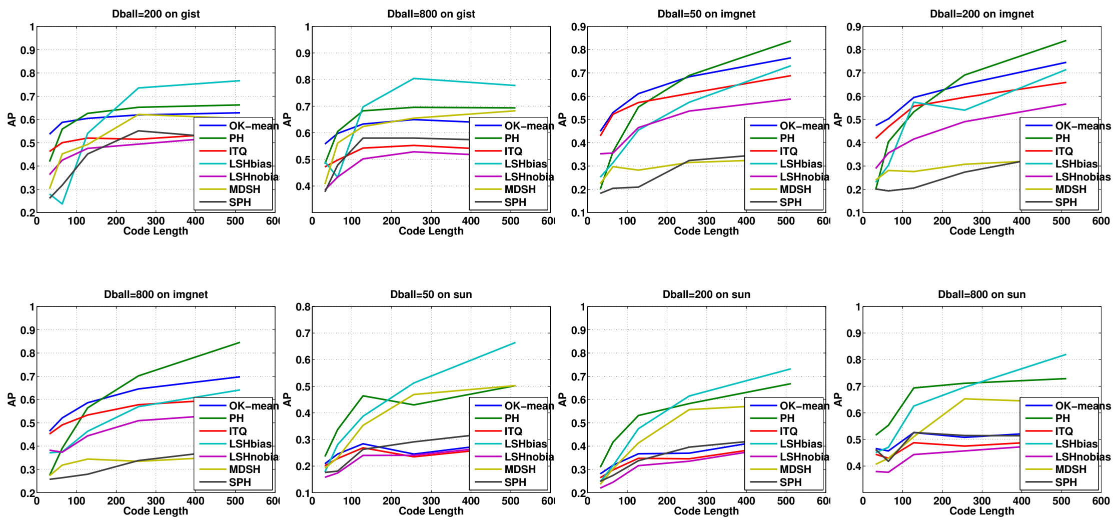


FIGURE 8.6: Average Precision vs Code length

of bits it performs worse than PH and LSHbias. As explained before, the feature space of those two datasets are less correlated compared to the SUN dataset.

Figure 8.6 shows the effect of the number of bits on the accuracy of retrieval. We vary the number of bits (Code length) by 32, 64, 128, 256 and 512. For each code length we measure the average precision by varying the radius size.

8.4.2.1 Predictability with p-stable perturbation

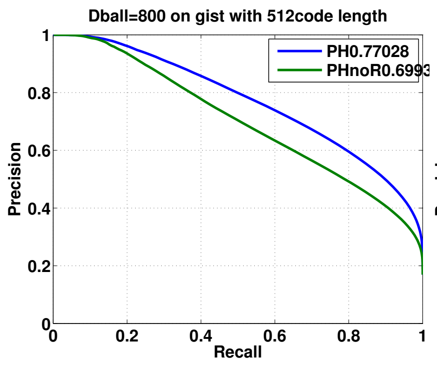
Since the optimization in ITQ is solved by a block coordinate descent algorithm, it may get stuck in local minima. In order to escape from the local minima, we again use the notion of predictability. As mentioned earlier, the bits produced by a hyperplane are predictable when the hyperplane has max-margin from data points. Equivalently, we can say that small changes to the predictable hyperplane should not change the binary values of the bits. This is because the hyperplanes are located within the gaps of the data and small changes will not flip a point to the other side of a hyperplane.

If we apply small perturbations to the normal vector of a predictable hyperplanes, the binary codes should be unchanged. If they do change, the perturbations can allow the solution to escape from a local minima. Formally, we are looking for a set of hyperplanes W (a $D \times K$ matrix), whose columns are normal vectors of hyperplanes, that produce the same binary codes for samples even when their coordinates are perturbed. With the perturbation, we can rewrite our criteria as follow:

$$\begin{aligned}
&\min_{B,R} \|B - W^T \hat{X}\|_F^2 \\
&s.t. \\
&\quad W = tR + (1-t)E, \\
&\quad RR^T = I,
\end{aligned} \tag{8.3}$$

where E is a $K \times K$ matrix of random values from the standard normal distribution and t is a balancing parameter for the amount of perturbation. We fix $t = 0.9$ in all of our experiments. One can tune this parameter to obtain better performance. This optimization is the same as ITQ but we apply small perturbations to the estimated rotation matrix at each iteration. The reason that the random perturbation values in E should be drawn from the standard normal distribution is because we need this mapping to preserve the neighbors under the ℓ_2-norm. As discussed in [173], the parameters of a random projection should come from a p-stable distribution to preserve ℓ_p-norm neighbors in the original feature space. The standard normal distribution is a 2-stable distribution, therefore it preserves neighbors under the ℓ_2-norm.

To measure the improvement due to random perturbations we performed an experiment without perturbation and compare with our full model (see Figure 8.7).



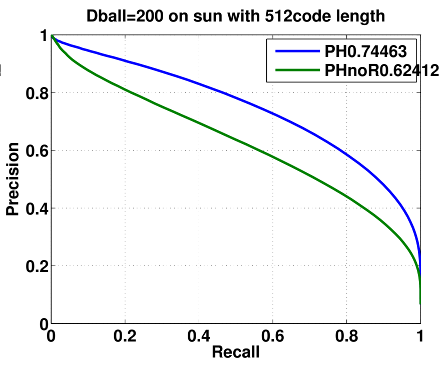


FIGURE 8.7: Impact of random perturbation in our method. We compare PHnoR (without random perturbation) with PH (random perturbation).

Chapter 9

Computationally Bounded Retrieval

9.1 Overview

Many computer vision problems can be formulated as a nearest neighbor search in some large dataset. When the data involved is high dimensional, doing this search efficiently becomes crucial but also extremely challenging. This is especially the case for the task of content based image retrieval, where efficiency is a requirement due to the high dimensionality of the data and the increase in size and availability of these databases.

The most common retrieval approaches are *tree based* and *hashing based* methods. In tree based approaches, the search space is embedded into a tree structure. At query time, the tree is traversed until a leaf node is reached, which points to a set of images that are similar to the query image. Then a final search is conducted over this set of images to find the nearest neighbor. Decision trees [176] and kd-trees[177, 178] are two such methods. When the number of dimensions grows, these conventional approaches often become less efficient, since the time or space requirements of these approaches often grows exponentially with the dimensionality.

Hashing based methods, on the other hand, reduce the number of dimensions involved with projections. This in turn removes the exponential dependence on dimensionality that trees suffer from. For instance, Locality Sensitive Hashing (LSH) uses random projections for the *encoding* step [38]. The images are indexed by hash tables, and at test time the query is encoded into an index that points to these hash tables. Typically the encoding step is the bottleneck in these image retrieval methods.

We focus primarily on image retrieval using binary codes [179]. In this approach, each image is encoded into a binary code such that the nearest neighbors in the corresponding Hamming space remain the same as the actual nearest neighbors in the original feature space. These binary codes can be used as hash keys to construct a hash table for real-time nearest neighbor

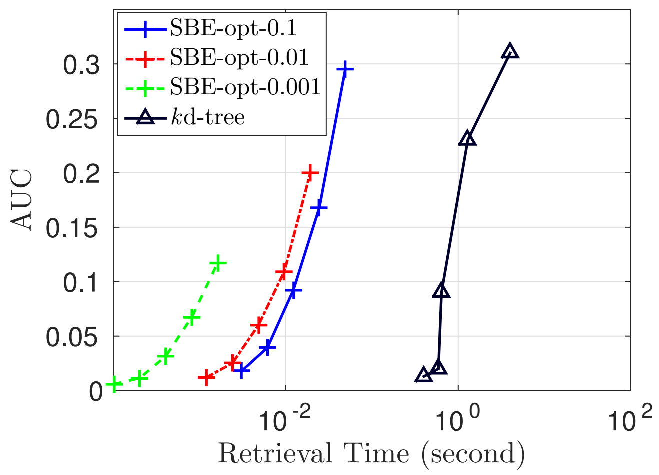


FIGURE 9.1: Comparison of the accuracy (area-under-the-curve) and computational cost of the proposed algorithm (SBE) with the kd-tree method for different sparsity values. The curves correspond to increasing code lengths for SBE and decreasing bucket size for kd-tree. Our method achieves the same accuracy as a kd-tree while being 100 times faster.

search [180]. Calculating the binary code, *i.e.* encoding, is in this case evaluating a mapping function f from some d-dimensional inputs to k-dimensional outputs. However if d and k are large, computational efficiency of f becomes a bottleneck at test time. Since the output is binary, this can be treated as a linear classification problem per bit. Even though there have been several efforts on making non-linear classification more efficient [181, 182], none of these methods are faster than linear classifiers, whereas making f more efficient would require sub-linear time complexity classifiers.

Typically, a single bit in a binary code is obtained by a dot product between the normal vector of a hyperplane and the feature vector, which implies $\mathcal{O}(dk)$ time complexity. In [183] a bilinear projection is employed to reduce the time complexity to $\mathcal{O}(d\sqrt{k})$. Recently, [184] proposed to use the columns of a circulant matrix as linear projections. This enables faster projection to generate k-bits ($k > 1$) by Fast Fourier Transform (FFT) which is $\mathcal{O}(d\log(d))$. We propose an optimization that learns a sparse projection to binary codes with a constraint on the computational budget.

We propose to find a mapping f that quantizes the data into binary values while: 1-*minimizing the quantization error*, and 2-*minimizing the computational cost of* f. We introduce two optimizations that employ ℓ_1-norm to constrain the computational operations in f based on the given sparsity rate α. In the first case we assume the binary codes are given and we only optimize for the mapping function f. In the second case we jointly optimize for binary codes and f. The joint optimization also introduces a new framework for learning binary codes, where the linear mapping function can be replaced by any arbitrary function.

We demonstrate our method in several image retrieval datasets (ImageNet, GIST, SUN) for nearest neighbor search. Our method achieves a high accuracy while having orders of magnitude less operations in comparison with the state-of-the-art methods on fast binary embedding. Surprisingly, our joint optimization even outperforms the baseline iterative quantization (ITQ) method [51] in terms of accuracy for some values of α. α provides a way of controlling the trade-off between speed and accuracy. Figure 9.1 shows that we achieve comparable accuracies to the kd-tree method using our algorithm, which requires only a fraction of the time a kd-tree requires.

9.2 Background

Most binary coding methods generate binary codes by projecting the data on linear hyperplanes that is followed by a binarization step (*e.g.* sign function). Given a data point $\mathbf{x} \in \mathbb{R}^d$, a k-bit binary code is generated by a hash function $h(\mathbf{x}) \in \{+1, -1\}^k$

$$h(\mathbf{x}) = \text{sign}(\mathbf{W}^{\mathsf{T}}\mathbf{x}); \tag{9.1}$$

where $\mathbf{W} \in \mathbb{R}^{d\times k}$ .There are two common approaches for generating the projection matrix $\mathbf{W}$; 1- **Random projection**: Elements of $\mathbf{W}$ are randomly chosen from a Gaussian distribution. This approach has been referred to as LSH in literature[38, 185, 186]. 2-**Data dependent**: The matrix $\mathbf{W}$ is learned from a set of training data to be optimized for a particular task (e.g. nearest neighbour retrieval or semantic search). There are many works on data dependent hashing [51, 187–191]. ITQ [51] is one of the popular methods that shows high accuracy in retrieval. It minimizes the quantization error over the training data after the dimensionality is reduced by PCA. In order to have uncorrelated bits in ITQ, the projection matrix is constrained to be a rotation matrix.

$$Q(\mathbf{R}, \mathbf{B}) = \min\|\mathbf{RPX} - \mathbf{B}\| \;\; s.t. \quad \mathbf{RR}^T = \mathbf{I} \tag{9.2}$$

where $\mathbf{P} \in \mathbb{R}^{k\times d}$ is the projection matrix from PCA and $\mathbf{X} \in \mathbb{R}^{d\times n}$ is the data matrix, such that each column in $\mathbf{X}$ is a d dimensional feature vector. $\mathbf{B} \in \{+1, -1\}^{k\times n}$ is the data in quantized form, and $\mathbf{R} \in \mathbb{R}^{k\times k}$ is the rotation matrix. The optimization is an iterative process over two steps; 1- fix $\mathbf{R}$ and solve for $\mathbf{B}$, and 2- fix $\mathbf{B}$ and solve for $\mathbf{R}$.

When $\mathbf{x}$ is a high dimensional vector, the computation time for this projection is prohibitive, since the computational complexity is $\mathcal{O}(dk)$. As discussed earlier, the computation cost for the binary projections can be very crucial in many applications. To overcome this cost [183] presents bilinear projection (BITQ) as an efficient way of generating binary codes.

$$h(\mathbf{x}) = \text{sign}(\mathbf{R}_1^T \mathbf{Z} \mathbf{R}_2) \tag{9.3}$$

where $\mathbf{Z} \in \mathbb{R}^{\sqrt{d} \times \sqrt{d}}$ is the reshaped version of data vector $\mathbf{x}$ ($\mathbf{vec}(\mathbf{Z}) = \mathbf{x}$) and $\mathbf{R}_1, \mathbf{R}_2 \in \mathbb{R}^{\sqrt{d} \times \sqrt{k}}$ are forced to be rotation matrices. This can be reformulated as follow:

$$h(\mathbf{x}) = \text{sign}((\mathbf{R}_1^T \otimes \mathbf{R}_2)\mathbf{x}) \tag{9.4}$$

where $\otimes$ is the tensor product. The computational complexity of bilinear projection in Equation 9.3 is $\mathcal{O}(d\sqrt{k} + \sqrt{d}k)$. Since $d >> k$, the dominant part of complexity would be $\mathcal{O}(d\sqrt{k})$. Similar to ITQ, there is an iterative process which can find the optimal $\mathbf{R}_1, \mathbf{R}_2$ to minimize quantization error.

In [184] Circulant Binary Embedding (CBE) is proposed for fast projection. The hashing function in CBE is in this form:

$$h(\mathbf{x}) = \text{sign}(\mathbf{C}\mathbf{D}\mathbf{x}) \tag{9.5}$$

where $\mathbf{C} \in \mathbb{R}^{d \times d}$ is a circulant matrix where the elements of the i^{th} column of $\mathbf{C}$ can be generated by one circular shift of $(i-1)^{th}$ column. and $\mathbf{D}$ is a diagonal matrix. The operation in Equation 9.5 can be implemented efficiently via Fast Fourier Transform (FFT) and the computational complexity of Equation 9.5 will be $\mathcal{O}(d \log(d))$.

In order to achieve faster projection in both BITQ and CBE, the projection matrix is forced to have a certain structure. In BITQ the projection has to come from a tensor product of two smaller rotation matrices and in CBE the projection matrix has to be a circulant matrix. These limitations on the structure of projection matrix can be seen as a type of regularization. In this work, we present a method that learns a sparse projection matrix $\mathbf{W}$ by directly regularizing the projection matrix using ℓ_1-norm. We can obtain very sparse projection matrices with this method, with sparsity rates of 0.1, 0.01 or even 0.001, which enables fast and accurate binary projection. Our results shows that we can reach comparable accuracies in retrieval with a sparsity rate of 0.01.

Figure 9.2 illustrates the computational cost of complexities dk, $d\log(d)$, $d\sqrt{k}$ and αdk with respect to the number of bits (k), where α is the sparsity rate of our proposed method. Evidently, αdk is more efficient with growing d.

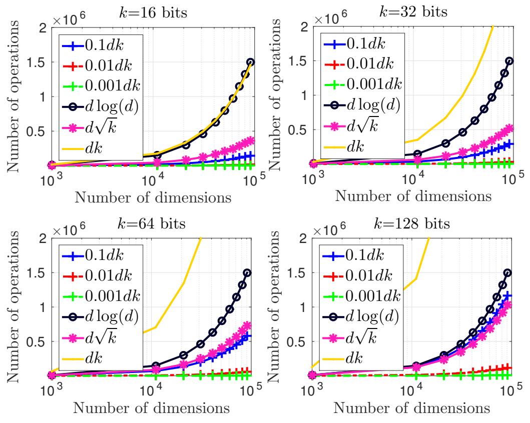


FIGURE 9.2: Complexity growth as a function of dimensions

9.3 Learning Computationally Constrained Binary Codes

Most machine learning problems can be formulated as learning of a prediction function $f_w : \mathcal{X} \to \mathcal{Y}$ which is a many-one mapping between some input space $\mathcal{X}$ and an output space $\mathcal{Y}$ that is parameterized by a parameter vector $\mathbf{w}$. Given a set of n training input-output pairs $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, the conventional empirical risk minimization approach for learning the optimal parameter vector $\mathbf{w}^*$ involves solving the following optimization problem:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(\mathbf{y}_i, f_w(\mathbf{x}_i)). \tag{9.6}$$

where $\ell(\mathbf{y}, \hat{\mathbf{y}})$ is a loss function that measures the discrepancy between the prediction $\hat{\mathbf{y}}$ and the ground truth output $\mathbf{y}$.

The choice of the representation of the prediction function f_w and the loss function ℓ results in different learning algorithms. For instance, a simple but popular representation for the prediction function for binary classification problems is:

$$f_w(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x}) \ , \tag{9.7}$$

where sign is element wise sign function $\{\text{sign} : \mathbb{R} \mapsto \{+1, -1\}\}$. In many large-scale computer vision and machine learning tasks, such as image labeling or classification, the inputs are high-dimensional, *i.e.* d is large. Many tasks also require multiple computations of the prediction function for a single input. For instance, this is the case for the problem of foreground-background segmentation where the task is to assign each pixel the foreground or background label by computing the predictor in a sliding window fashion. Furthermore, one may need this computation to work in real time: more than 20 frame per second. In these case, even computing the simple linear mapping defined in Equation 9.7 may become computationally expensive because of the large number of multiplication required to compute the dot-product between $\mathbf{w}$ and $\mathbf{x}$.

In the case of binary codes learning the parameter vector $\mathbf{w}$ is replaced by a parameter matrix $\mathbf{W}$. In this paper, we focus on binary code predictors that tie themselves to a fixed computational budget. Specifically, we consider the case where there is a limit on the number of arithmetic operations that can be performed at test time. More formally, we want to solve the computation-bounded risk minimization problem that is defined as:

$$\underset{\mathbf{W},\mathbf{b}}{\text{argmin}} \quad \textstyle\sum_{i=1}^{n} \ell(\mathbf{b}_i, f_w(\mathbf{x}_i)) \tag{9.8}$$
$$st. \qquad \tau(f_w) \leq l \tag{9.9}$$

where $\mathbf{b}_i \in \{-1, 1\}^k$ is the desired binary code for $\mathbf{x}_i$ and τ measures the computation complexity of evaluating the prediction and l is the required computational bound. For the case of predictor defined in Equation 9.7, the above problem translates to

$$\underset{\mathbf{W},\mathbf{b}}{\text{argmin}} \quad \textstyle\sum_{i=1}^{n} \ell(\mathbf{b}_i, \text{sign}(\mathbf{W}^{\mathsf{T}}\mathbf{x}_i)) \tag{9.10}$$
$$st. \qquad ||\mathbf{w}||_0 \leq l \tag{9.11}$$

where $||\mathbf{w}||_0$ denotes the ℓ_0 norm that counts the number of non-zero components of $\mathbf{W}$ since only these many multiplications are needed to evaluate the function.

In contrast to the classification problem, there are no ground truth labels for our task of binary code learning. In fact, the classifier and the labels should both be estimated, which is a complicated task. In the next subsection we explain how to learn an optimal computationally bounded function when the desired binary codes are given. In subsection 9.3.2, we propose a joint optimization that solves for both binary codes and the mapping function jointly.

9.3.1 Sparse Projection When Binary Codes Are Given

To improve the computation cost of f, a relatively straightforward idea is to make the matrix $\mathbf{W}$ sparse. When $||\mathbf{W}||_0 \leq l$, *i.e.* when number of non-zero entries of $\mathbf{W}$ is at most l, then clearly f can be computed in $\mathcal{O}(l)$. However, directly solving for ℓ_0-norm is intractable. Therefore, sparsity is often incorporated by introducing an ℓ_1 penalty on the parameter matrix $\mathbf{W}$ followed by thresholding.

In our approach we minimize the quantization error in the original feature space, which is in essence similar to ITQ which minimizes the quantization error in the PCA space. However, we force the $\mathbf{W}$ to be a sparse matrix.

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}}\{\|\mathbf{W}^\mathsf{T}\mathbf{X} - \mathbf{B}\|_F + \lambda|\mathbf{W}|_{\ell_1}\} \tag{9.12}$$

Here $|.|_{\ell_1}$ operator on a matrix is equivalent to sum of the absolute values of the elements in that matrix. We assume that the binary codes $\mathbf{B}$ are computed as a part of training via one of the best binary coding methods (*e.g.* ITQ). It can be shown that the optimal solution for $\mathbf{W}$ can be computed independently for each column of $\mathbf{W}$. Given the matrix $\mathbf{B}$ from the output of ITQ (Equation 9.2), we can rewrite Equation 9.12 as follows:

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}}\{\sum_{i=1}^{k} \|\mathbf{w}^{i\mathsf{T}}\mathbf{X} - \mathbf{b}^i\|_{\ell_2} + \lambda|\mathbf{w}^i|_{\ell_1}\} \tag{9.13}$$

where $\mathbf{w}^i$ and $\mathbf{b}^i$ are i^{th} row of $\mathbf{W}^\mathsf{T}$ and $\mathbf{B}$ respectively, and λ is a trade-off parameter. Equation 9.13 is a minimization over sum of k positive elements such that the parameters in each element is independent. Therefore, the global minimum is equivalent to the sum of the minimum values in each element:

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}}\{\sum_{i=1}^{k} \min_{\mathbf{w}^i}\{\|\mathbf{w}^{i\mathsf{T}}\mathbf{X} - \mathbf{b}^i\|_{\ell_2} + \lambda|\mathbf{w}^i|_{\ell_1}\}\} \tag{9.14}$$

$$\mathbf{w}^{*i} = \underset{\mathbf{w}^i}{\operatorname{argmin}}\{\|\mathbf{w}^{i\mathsf{T}}\mathbf{X} - \mathbf{b}^i\|_{\ell_2} + \lambda|\mathbf{w}^i|_{\ell_1}\} \tag{9.15}$$

$$\mathbf{W}^* = \begin{bmatrix} \mathbf{w}^{*1} & \mathbf{w}^{*2} & \dots & \mathbf{w}^{*k} \end{bmatrix} \tag{9.16}$$

Intuitively, elements in each row of $\mathbf{B}$ can be considered a binary class label for the columns of $\mathbf{X}$. Similarly, each column of $\mathbf{W}$ can be considered a classifier that predicts the binary labels. Therefore, we can reformulate the optimization for i^{th} column of $\mathbf{W}$ as a max-margin ℓ_1 regularized linear classifier:

$$\begin{aligned} &\underset{\mathbf{w}^i}{\operatorname{argmin}} && |\mathbf{w}^i|_{\ell_1} + C\sum_{j=1}^{n}\xi_j \\ &\text{subject to} && \mathbf{b}^i(j)(\mathbf{w}^{i\mathsf{T}}\mathbf{x}_j) \geq 1 - \xi_j,\ j = 1, \dots, n. \end{aligned} \tag{9.17}$$

Here $\mathbf{x}_j$ is the j^{th} column of $\mathbf{X}$. We solve this optimization efficiently by using LibLinear [192]. After obtaining the optimized $\mathbf{w}$, we zero out the dimensions with small absolute values to reach the sparsity rate of α. This is done simply by sorting the absolute values in $\mathbf{w}$ and pick a threshold based on the given sparsity rate (*e.g.* if the sparsity rate is 0.1, we zero out 90% of the smallest values). Computational complexity of obtaining a binary code for a point $\mathbf{x}$ is $\mathcal{O}(\alpha dk)$ where α is the sparsity rate of $\mathbf{W}$. Our experimental results show that even for very sparse projections with $\alpha = 0.01$, the accuracy drop is insignificant. As discussed earlier, for high dimensional data, generating short binary codes will lead to much less number of operations than BITQ and CBE as explained on Figure 9.2. In the next subsection we discuss a case, where the binary codes are not given during the training phase and we should optimize for both $\mathbf{B}$ and $\mathbf{W}$.

9.3.2 Joint Optimization

So far, binary codes were supplied in the training phase. These binary codes could be obtained from the output of any compelling binary coding method (*e.g.* ITQ). As there is no guarantee that the exact codes can be reconstructed by sparse mapping, we need to incorporate the search for binary codes into the main objective. Similar to the Equation 9.12, we aim to minimize the quantization error while maintaining the low ℓ_1-norm. In contrast to Equation 9.12, $\mathbf{B}$ is an unknown variable.

$$(\mathbf{W}^*, \mathbf{B}^*) = \underset{\mathbf{W},\mathbf{B}}{\operatorname{argmin}}\{\|\mathbf{W}^\mathsf{T}\mathbf{X} - \mathbf{B}\|_F + \lambda|\mathbf{W}|_{\ell_1}\} \tag{9.18}$$

To solve this optimization problem, one might consider an EM like iterative update of variables; 1- fix $\mathbf{B}$ and solve for $\mathbf{W}$ as described in Section 9.3.1, 2- fix $\mathbf{W}$ and solve for $\mathbf{B}$ which is $\mathbf{B} = \operatorname{sign}(\mathbf{W}^\mathsf{T}\mathbf{X})$. However, solving this optimization does not entail a desirable solution. The reason is, there is no control over dependency of bits of binary codes, *i.e.* there is no constraint on $\mathbf{B}$. Usually, low correlation can be achieved by an orthogonality constraint over the parameters of the mapping function. In our case, it would be equivalent to have $\mathbf{W}^\mathsf{T}\mathbf{W} = \mathbf{I}$ as a constraint in the objective. However, having this constraint along with the ℓ_1 penalty complicates the optimization problem, since orthogonality and sparsity of the mapping function are two competing constraints on minimizing the quantization error. Therefore we employ the iterative optimization procedure explained above, but this time considering only one of the constraints at each step.

To solve the optimization of Equation 9.18, we replace the matrix $\mathbf{B}$ with an explicit sign function of an orthogonal projection of the data as follows:

$$\begin{aligned}(\mathbf{W}^*, \mathbf{P}^*) =&\underset{\mathbf{W},\mathbf{P}}{\operatorname{argmin}}\{\|\mathbf{W}^\mathsf{T}\mathbf{X} - \operatorname{sign}(\mathbf{P}^\mathsf{T}\mathbf{X})\|_F + \lambda|\mathbf{W}|_{\ell_1}\} \\ &s.t. \qquad \mathbf{P}^\mathsf{T}\mathbf{P} = \mathbf{I}\end{aligned} \tag{9.19}$$

where $\mathbf{P} \in \mathbb{R}^{d\times k}$ is an orthogonal matrix. This orthogonal projection ensures the low correlation between the bits, and also in contrast to Equation 9.18, it provides an update for binary codes which is not directly dependent on $\mathbf{W}$. Again, we employ the iterative two steps block-coordinate descend technique, where we iterate between solving for $\mathbf{W}$ and $\mathbf{P}$. When $\mathbf{P}$ is fixed, the problem would be the same as what we had in Section 9.3.1; the optimal $\mathbf{W}$ can be obtained via linear ℓ_1-SVMs. When $\mathbf{W}$ is fixed, the remaining optimization is as follows (let $\mathbf{M} = \mathbf{W}^\mathsf{T}\mathbf{X}$):

$$\begin{aligned} \mathbf{P}^* =& \underset{\mathbf{P}}{\text{argmin}}\{\|\mathbf{M} - \text{sign}(\mathbf{P}^\mathsf{T}\mathbf{X})\|_F\} \\ s.t. \quad & \mathbf{P}^\mathsf{T}\mathbf{P} = \mathbf{I} \end{aligned} \tag{9.20}$$

The optimized solution for $\mathbf{P}$ in Equation 9.20 is not unique. Let's define $\mathcal{P}$ as the optimal solution set for $\mathbf{P}^*$. Solving this optimization is intractable due to the non-linearity of the sign function. Instead of solving 9.20, we claim that using the following optimization gives an optimal $\mathbf{P}$:

$$\begin{aligned} \mathbf{P}^{**} =& \underset{\mathbf{P}}{\text{argmin}}\{\|\,\text{sign}(\mathbf{M}) - \mathbf{P}^\mathsf{T}\mathbf{X}\|_F\} \\ s.t. \quad & \mathbf{P}^\mathsf{T}\mathbf{P} = \mathbf{I} \end{aligned} \tag{9.21}$$

The above optimization(9.21) is an orthogonal procrustes problem and has a closed form solution. $\mathbf{P}^{**} = \mathbf{V}_{(1:k,:)}{}^\mathsf{T}\mathbf{U}$ where $(\mathbf{U}, \mathbf{D}, \mathbf{V}) = \text{svd}(\text{sign}(\mathbf{M})\mathbf{X}^\mathsf{T})$ [1]. It can be shown that $\mathbf{P}^{**} \in \mathcal{P}$. Lets consider a single element in $\mathbf{M}$ as $\mathbf{M}_{(i,j)}$. In 9.20 we would have :

$$\begin{aligned} \mathbf{P}^*_{(:,i)} =& \underset{\mathbf{P}_{(:,i)}}{\text{argmin}}\{\|\mathbf{M}_{(i,j)} - \text{sign}(\mathbf{P}_{(:,i)}{}^\mathsf{T}\mathbf{X}_{(:,j)})\|\} \\ s.t. \quad & \mathbf{P}_{(:,i)}{}^\mathsf{T}\mathbf{P}_{(:,i)} = 1 \end{aligned} \tag{9.22}$$

if $\mathbf{M}_{(i,j)} \geq 0$, the optimal solution set

$$\mathcal{P}_{(:,i)} = \{\forall \mathbf{p} \in \mathbb{R}^d |\ \mathbf{p}^\mathsf{T}\mathbf{X}_{(:,j)} \geq 0,\ \mathbf{p}^\mathsf{T}\mathbf{p} = 1\} \tag{9.23}$$

The optimal solution from Equation 9.21 is $\mathbf{P}^{**}_{(:,i)} = \text{sign}(\mathbf{M}_{(i,j)})\frac{\mathbf{X}_{(:,j)}}{\|\mathbf{X}_{(:,j)}\|}$. Replacing $\mathbf{P}^{**}_{(:,i)}$ with $\mathbf{p}$ in Equation 9.23 proves that $\mathbf{P}^{**}_{(:,i)} \in \mathcal{P}_{(:,i)}$. Analogously, we can prove the same solution when $\mathbf{M}_{(i,j)} < 0$. Therefore, an optimal solution for 9.21 would also be an optimal solution for Equation 9.20.

In Algorithm 6 we show all the steps in our joint optimization for binary code learning. The joint optimization has an advantage over original ITQ formulation: in contrast to ITQ, the mapping function is directly learned over the original feature space, not on the PCA reduced space. Our experiments verify that the proposed method can outperform ITQ while being at least ten times faster.

In the next section we show an extensive set of experiments to evaluate all the parts of our proposed method in comparison to the other state-of-the-art methods.

[1]The (:) notation is the same as being used in MATLAB

Algorithm 6 Sparse Binary Embedding

```
Input: $\mathbf{X} \in \mathbb{R}^{d\times n}$, $k$(number of bits), $\lambda, \alpha$, $niter$.
Output: $\mathbf{W} \in \mathbb{R}^{d\times k}$, $\mathbf{B} \in \{1, -1\}^{k\times n}$.
 1: $\mathbf{P} \leftarrow$ $random\ orthogonal\ matrix^{d\times k}$
 2: repeat
 3:     $\mathbf{L} \leftarrow \text{sign}(\mathbf{P}^{\mathsf{T}}\mathbf{X})$
 4:     $\mathbf{W} \leftarrow \ell_1 - \text{LinSVMs}(\text{train data} = \mathbf{X}, \text{train labels} = \mathbf{L}, \text{hyperparameter} = 1 - \lambda)$
 5:     $\mathbf{M} \leftarrow \mathbf{W}^{\mathsf{T}}\mathbf{X}$
 6:     $(\mathbf{U}, \mathbf{D}, \mathbf{V}) \leftarrow \text{svd}(\text{sign}(\mathbf{M})\mathbf{X}^{\mathsf{T}})$
 7:     $\mathbf{P} \leftarrow {\mathbf{V}_{(1:k,:)}}^{\mathsf{T}}\mathbf{U}$
 8:     $Objective \leftarrow \|\mathbf{M} - \mathbf{L}\|$
 9: until convergence on $Objective$
10: $\mathbf{W} \leftarrow$ zero out values in columns of$\mathbf{W}$ to reach the sparsity rate $\alpha$
11: $\mathbf{B} \leftarrow \text{sign}(\mathbf{W}^{\mathsf{T}}\mathbf{X})$
```

9.4 Experiments

In this section, we measure the accuracy and computational cost of our method on different datasets for image retrieval. The sparsity of our mapping function also proves to be beneficial when the data is noisy. Therefore, we also present an evaluation on retrieval from noisy data.

9.4.1 Experimental Setting

Datasets: We consider three datasets in our experiments; ImageNet[193], GIST-1M[194], and SUN14k[195]. Each dataset is randomly partitioned into three sets: *train, query* and *base*. The parameters (mapping functions) are trained on the training set and we use each sample in the query set to find its nearest neighbor in the base set. ImageNet includes 1000 categories of images and has 1281167 images in total. We specifically used the ISLVRC2012 benchmark dataset. 100K, 1K, 1180K samples were picked for training, query and base respectively. We used CNN features extracted by Caffe [196] with 4096 dimensions. Gist1M [194] contains 1M images and their 960 dimensional GIST features. This dataset has frequently been used for approximate nearest neighbor search in the computer vision community. It has a standard partitioning of training(500K), query(100) and base(1M) sets. SUN14K[195] is a dataset of 700 categories of images, which is mainly used for scene recognition. We used this dataset because of its high dimensional features (19080 dimensions). This dataset has been used for attribute based recognition and dual-view hashing by Patterson and Hays [195] and Rastegari et al. [197]. It has 14K images and the visual features extracted by Patterson and Hays [195], which is a concatenation of GIST, PHOG and BoW. The reason that we chose these datasets are: **ImageNet:** to evaluate our method on the state-of-the-art ConvNet (CNN) features that showed great potential for recognition[198], **GIST1M:** to evaluate the accuracy of our method on a standard image retrieval dataset, and **SUN14K:** because of its feature set that allows us to evaluate the efficiency of our method on very high dimensional spaces. In order to have a fair

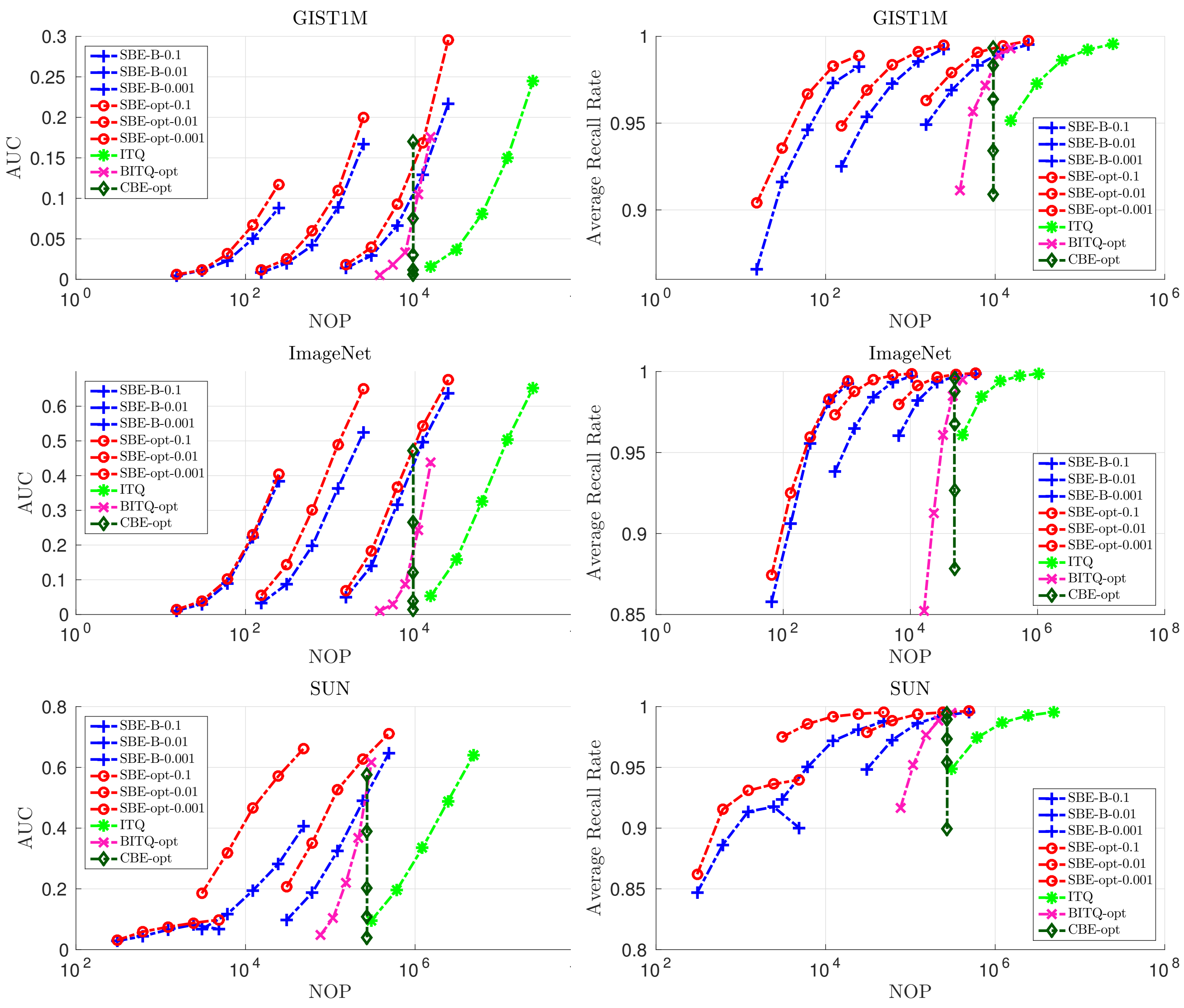


FIGURE 9.3: **Number of operations vs. Accuracy (AUC, Recall):** In this plot each curve represents a method and each point on the curve corresponds to a code length (number of bits) which change from left to right as follows: [16, 32 64, 128, 256]

comparison with other baselines, it is very important to pre-process the data by mean centering and normalizing them to unit hyper-sphere [184, 199].

Methods: To best of our knowledge, there are two other works that focus on fast binary embedding [183, 184]. Similar to [184], we use ITQ [51] as a baseline method which is not an efficient method but it gives the best accuracy among other binary embedding techniques [199]. Circulant Binary Embedding (**CBE**)[184] and Bilinear Iterative Quantization (**BITQ**) are used as competitor methods in terms of efficiency and accuracy. These methods have two versions: one with random paramaters (**CBE-rand**, **BITQ-rand**) and another one with optimized parameters (**CBE-opt, BITQ-opt**). Here we only use the optimized versions for comparison. Our method is abbreviated by (**SBE**) which stands for Sparse Binary Embedding. We distinguish between the optimization in Section 9.3.1 by (**SBE-B**) and the joint optimization in Section 9.3.2 by (**SBE-opt**). In most experiments, we measure our method with a sparsity rate of $\alpha = 0.01$, however, we also explore the sparsity rates 0.1 and 0.001 in Section 9.4.4. For these comparisons, we used the ITQ, BITQ and CBE software that is available online.

9.4.2 Nearest Neighbor Retrieval

In this section, we demonstrate the accuracy of the nearest neighbor retrieval task on the benchmark datasets. We present both recall rate and precision in the form of area-under-curve (AUC) (Subsection 9.4.4). Similar to [184], the steps of this experiment for each method is as follows: 1- Learn a model using the training set. 2- Create the binary codes for all the samples in the base set. 3-Pick one sample from query set and find its K-nearest neighbors in the base set by using Euclidean distance in the original features space and consider this as ground-truth nearest neighbor set. 4- Generate the binary code for the query sample and retrieve N nearest neighbors from base set by calculating the Hamming distances on the binary codes. 5- Compare the retrieval from binary codes with the ground-truth retrieval and measure the recall rates. This process is repeated over all samples in the query set and the average recall rate is computed. By varying N at $K = 100$, we measure recall rate at different amount of retrieval tasks and plot a curve using these values. Figure 9.4, 9.5 and 9.6 show the recall rate as we change the number of retrieval in different code length (bit). The curve of SBE-opt is on top of all the competitor methods most of the times. It even out-performs the method **ITQ**, which is the most accurate out of all the baseline methods.

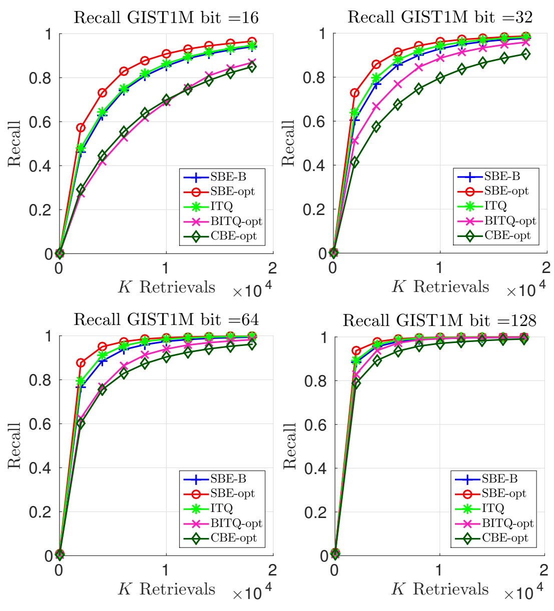


FIGURE 9.4: Recall curves for GIST1M dataset

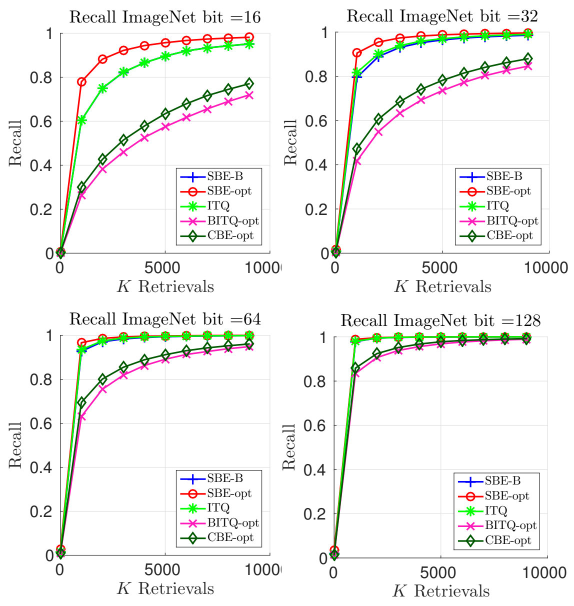


FIGURE 9.5: Recall curves ImgeNet dataset

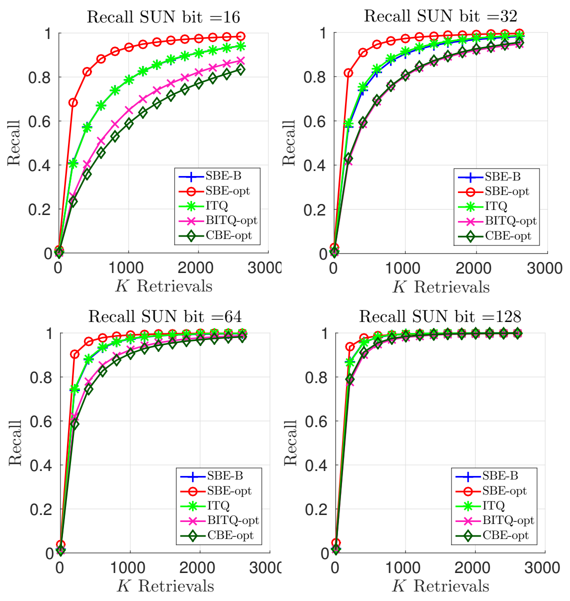


FIGURE 9.6: Recall curves for SUN dataset

9.4.3 Analysis of the objective in joint optimization

Here we present a comparison of the optimization functions proposed in Section 9.3.2. We refer to Equation 9.18 as the *Naive Update*. This optimization produces highly correlated bits which may have high quantization error. The new formulation presented in Equation 9.19 that handles the orthogonality constraints is referred to as *Orthogonal Update*. In Figure 9.7, the first row demonstrates the correlation between the bits of the binary codes generated from the two optimization functions. As it can be seen, in contrast to Orthogonal Update, the bits resulting from Naive Update are highly correlated. In Figure 9.7, the second row shows the quantization error after each iteration. The quantization error, as expected, goes up for Naive Update but for the Orthogonal Update it goes down.

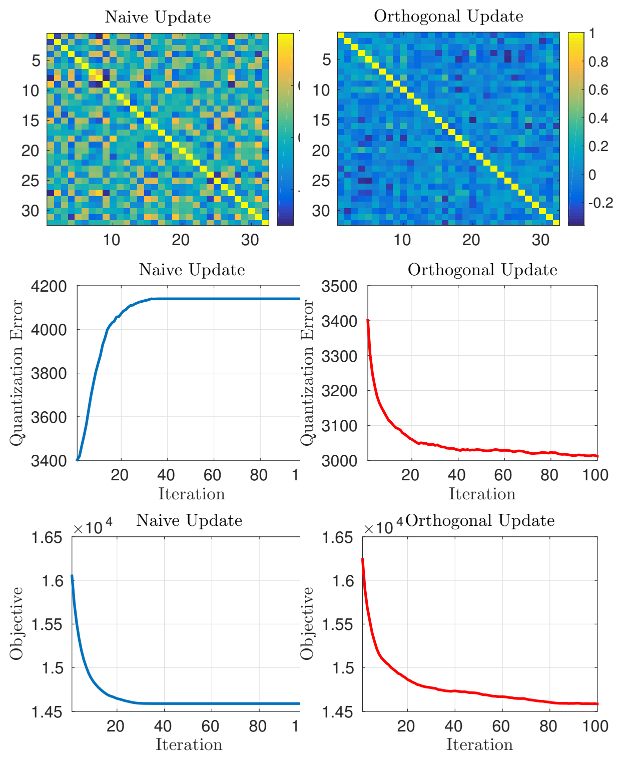


FIGURE 9.7: Analysis on the optimization of the objective function on sparse binary coding

Another claim was that Naive Update should converge much faster than the Orthogonal Update, since the latter has two competing constraints. This is shown in Figure 9.7, third row.

9.4.4 Coding Efficiency vs. Accuracy

In this section we compare the efficiency of our method with other competitor methods. In order to present a fair comparison, instead of reporting the running time, we report the number of arithmetic operations (**NOP**) used for each method. Figure 9.3 shows an evaluation over NOP and accuracy which is measured by AUC and average recall rate. Each curve in Figure 9.3 depict one method and each point on the curve corresponds to a code length (number of bits) which is in the range of [16, 32, 64, 128, 256]. Evidently, in ImageNet SBE can achieve comparable accuracy with 128 bits while being 1000 times faster than other methods. The dark green curve corresponds to CBE is growing vertically in the plot. This is because the complexity of CBE $d\log(d)$ is independent of the number of bits and only depends on the number of dimensions in the original feature space.

9.4.5 Retrieval Running Time Analysis

Since the main application for our fast binary encoding method is retrieval, we compared it with kd-tree as a baseline for a nearest neighbor retrieval task. We used multiple index hashing as a retrieval technique on top of our binary codes. For each eight consequent bits, we generate a hash table to index the samples in the database. At query time, we perform $\frac{b}{8}$ look-ups. b is the number of bits in our binary code. We score the collected indexes by counting the number of time that they have been retrieved from the hash tables. Then, we pick the k indices with the highest score as the k nearest neighbors. We vary the number bits by [16, 32, 64, 128, 256] and compute the AUC and running time. In kd-tree we vary the bucket size by [2^2, 2^4, 2^8, 2^{10}, 2^{12}] and compute the AUC and running time. The results on GIST1M are depicted in Figure 9.1. We used the built in kd-tree toolbox in MATLAB for this comparison.

9.4.6 Retrieval from Noisy Data

The sparsity of the projection matrix in our method makes it possible to have accurate retrieval with noisy data. In Figure 9.8 we show the performance of our method in comparison with others on the noisy data in ImageNet. Noisy data is created by randomly selecting some dimensions of each sample and change their values to either 0 , 1 or -1. As it can be seen our method can reach higher accuracy compared to ITQ with high noise ratio.

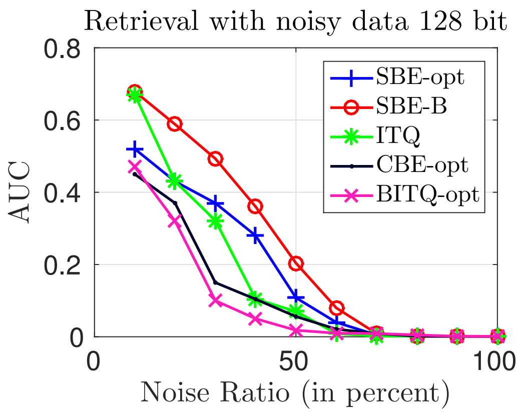


FIGURE 9.8: Retrieval with noisy feature on ImageNet

Chapter 10

On Large-Scale Retrieval: Binary or n-ary Coding?

10.1 Overview

Large-scale retrieval has attracted a growing attention in recent years due to the need for image search based on visual content and the availability of large-scale datasets. In this chapter we focuses on the problem of approximate nearest neighbor (ANN) search for large-scale retrieval.

Approaches for solving this problem generally fall into two subcategories; **Fast Distance Estimation** [194, 200] and **Fast Subset Indexing**[38, 176, 180, 185, 186]. Fast Distance Estimation methods reduce computation cost by approximating the distance function. Distance computation is very expensive in high dimensional feature spaces. On the other hand, Fast Subset Indexing methods reduce the cost by constraining the search space for a query to a subset of the dataset instead of the whole dataset.

A general technique for ANN search (both Fast Distance Estimation and Fast Subset Indexing) is to discritize the feature space into K regions. Different coding methods can be used for this purpose. One of the classic methods is one-hot encoding using K-means. K-means is a classic quantization technique that quantizes data into K regions (clusters). Data is coded using a K bit binary code, in which only one bit is one (representing the appropriate cluster) . Although K-means works well for small values of K, it becomes intractable for large K.

An alternative method to one-hot encoding using K-means is binary coding. One can code the K clusters with $m = log_2(K)$[1] dimensional binary codes by relaxing the one-hot encoding constraint and allowing multiple bits to be one. This is equivalent to partitioning the space

[1]Without loss of generality, we assume that K is selected such that m is a natural number.

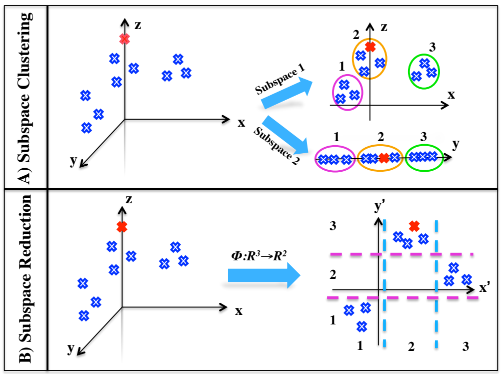


FIGURE 10.1: Difference between Subspace Clustering (Part A) and Subspace Reduction (Part B) for generating n-ary codes. The example shows the simple case of generating 2-dimensional 3-ary codes. In Subspace Clustering, data is clustered into 3 clusters in the two defined subspaces (e.g. the code for the red cross is $[2,2]^T$). In Subspace Reduction, data is transformed into a two dimensional space and each dimension is discretized into 3 bins (e.g. the code for the red cross is $[2,3]^T$).

into two regions m times. Many binary coding methods have been designed to address this problem[38, 51, 188, 190]. While binary coding is more scalable, it has a high reconstruction error.

Binary coding can be relaxed by allowing each dimension to be n-ary instead of binary (i.e. take on integer values between 1 and n). In this case, K clusters can be coded with $m = log_n(K)$ dimensional n-ary codes. We introduce a new categorization for methods that generate n-ary codes. We explore two general approaches to generate m-dimensional n-ary codes: 1- **Subspace Clustering**: In this approach, the original feature space is divided into m subspaces and each subspace is quantized into n clusters. 2- **Subspace Reduction:** Here, the dimensionality of the original feature space is reduced to m and each dimension is quantized into n bins. Figure 10.1 illustrates these approaches. Multi-dimensional quantization methods (e.g. PQ, CK-means)[194, 200, 201] adopt the first approach to perform n-ary coding. They solve the problem for any m and n (including $n = 2$ which leads to binary coding based on the first approach). On the other hand, many binary coding methods(e.g. ITQ, LSH) [38, 51, 188, 190] are instances of the second approach. However they are limited to the case where $n = 2$.

Most recent papers on quantization [200, 201], compared their proposed methods with binary coding methods only with respect to *Distance Estimation* (i.e. typically employing exhaustive search over the data, where the approximated distance is mimicking the ordering of images based on Euclidean distance in the original feature space). This leaves the question of ”which binary or n-ary coding performs better for ANN search using *Subset Indexing*?” unanswered.

The contributions of this approach are twofold. *First*, a new general approach for multi-dimensional n-ary coding is introduced. Based on that, Linear Subspace Quantization (LSQ) is proposed as a new multi-dimensional n-ary encoder. Unlike previously proposed n-ary encoders in which the Euclidean distance between n-ary codes is not preserved, the distances in LSQ coded space correlate with the Euclidean distance in the original space. As a result, the codes can be used directly for learning tasks. Furthermore, LSQ does not make the restrictive assumption of dividing space into independent subspaces, which is common in n-ary encoders. Experiments show that LSQ outperforms such encoders. *Second*, it is shown that n-ary coding does not always outperform binary coding in retrieval. We show that binary coding works better when Subset Indexing is used and present an explanation based on the two approaches to coding. To the best of our knowledge this has not been identified previously. However, it is very important for large-scale retrieval.

The rest of this chapter is organized as follows. In section 10.2, the general formulation for both Subspace Clustering and Subspace Reduction is presented. Additionally, the LSQ coding method is described and its relation to other methods and its properties are discussed. Section 10.3, describes the ways n-ary and binary coding methods can be exploited in combination with distance estimation and subset indexing to reduce search cost in retrieval. Experiments are reported in Section 10.4.

10.2 n-ary Coding

ANN search methods discretize the feature space into a set of disjoint regions. n-ary coding can be used for this purpose. An n-ary code of length m is defined by an m-dimensional vector in $\{1, 2, \dots, n\}^m$. The goal is to transform data into m-dimensional n-ary codes that reconstruct the original data accurately.

First, consider constructing a *one*-dimensional n-ary code. A common objective for quantization methods is to minimize the reconstruction error, referred to as *quantization error*. In other words, given a set of data points $\mathbf{X} \in \mathbb{R}^{D \times N}$ where each column is a data point $\mathbf{x} \in \mathbb{R}^D$, the quantization objective can be expressed as:

$$\min_{Q}\{\|\mathbf{X} - Q(\mathbf{X})\|_F^2\} \tag{10.1}$$

where Q maps a vector $\mathbf{x}$ (column in $\mathbf{X}$) into one element of a finite set of vectors $\mathbf{C} = \{\mathbf{c}_1, \mathbf{c}_2, \dots \mathbf{c}_n\}$ in $\mathbb{R}^D$ referred to as a codebook. The index of the codebook vector assigned to a data point is its one-dimensional n-ary code. K-means optimizes this objective when the size of the codebook is equal to K.

Using the one-hot encoding notation, the optimization in 10.1 can be written as follows:

$$\min_{\mathbf{B},\mathbf{C}}\{\|\mathbf{X} - \mathbf{C}\mathbf{B}\|_F^2\} \tag{10.2}$$

where $\mathbf{B} \in \{0,1\}^{K \times N}$ is a binary matrix in which each column is a 1-way selector - all of its elements but one are zero.

In order to generalize one-dimensional n-ary codes to m-dimensional codes, we explore two approaches: *Subspace Clustering* and *Subspace Reduction*. Although the former has been explored in the literature, the latter has not. Without loss of generality, we assume that the data are mean centered (i.e. $mean(\mathbf{X}) = 0_{D\times1}$) and scaled to $[-1, 1]$ by mapping the data to the unit hyper-sphere. In [51, 199], it is shown that it is very beneficial to normalize the data to the unit hyper-sphere.

10.2.1 Subspace Clustering

Here, to generate m-dimensional n-ary codes, the original feature space is divided into m subspaces and each subspace is discretized into n regions. To this end, in 10.2, the number of clusters can be set to $K = n^m$ and the selector can be allowed to include m non-zero elements as follows:

$$\min_{\mathbf{B},\mathbf{C}}\{\|\mathbf{X} - \left[\begin{array}{c|c|c|c} \mathbf{C}_1 & \mathbf{C}_2 & \ldots & \mathbf{C}_m \end{array}\right] \left[\begin{array}{c} \mathbf{B}_1 \\ \hline \mathbf{B}_2 \\ \hline \vdots \\ \hline \mathbf{B}_m \end{array}\right] \|_F^2\} \tag{10.3}$$

Here $\mathbf{C}_i$ and $\mathbf{B}_i$ are the codebook and its related one-hot encoding in the i'th subspace. In general, the optimization of 10.3 is intractable. As a result, Product Quantization[194] and Cartesian K-means [200] solve a constrained version of this problem where the subspaces created by the $\mathbf{C}_i$s are orthogonal. In other words, $\{\forall i, j|\ i \neq j\ \ {\mathbf{C}_i}^\mathsf{T}\mathbf{C}_j = 0_{n\times n}\}$. Intuitively, the original space is divided into m independent subspaces and each is clustered into n regions. We next present another approach to n-ary coding in which no such constraint is imposed.

10.2.2 Subspace Reduction

Subspace Reduction maps the data into an m-dimensional space and discretizes each dimension into n bins. The goal is to perform this discretization to minimize the reconstruction error in the original space. Formally, the optimization problem can be written as follows:

$$\min_{f,\tilde{f}}\{\|\mathbf{X} - \tilde{f}(q_n(f(\mathbf{X})))\|_F^2 + \lambda R(\tilde{f})\} \tag{10.4}$$

where $f : \mathbb{R}^D \mapsto \mathbb{R}^m$ is the *mapping function* and is applied to each column of $\mathbf{X}$, $\tilde{f} : \mathbb{R}^m \mapsto \mathbb{R}^D$, $(m \leq D)$ is the *reconstruction function* which projects the data back to the D dimensional space in which the reconstruction error is computed. In order to prevent overfitting, the reconstruction function must be regularized. R is a regularizing function that limits the variations in $\tilde{f}$, λ is a parameter controlling the amount of regularization, and q_n is a uniform quantizer that is applied to each element of its input and is defined as:

$$q_n(x) = \begin{cases} \theta_n(1) & x < \frac{\theta_n(1)+\theta_n(2)}{2} \\ \theta_n(2) & \frac{\theta_n(1)+\theta_n(2)}{2} \leq x < \frac{\theta_n(2)+\theta_n(3)}{2} \\ \vdots & \\ \theta_n(n) & \frac{\theta_n(n-1)+\theta_n(n)}{2} \leq x \end{cases} \tag{10.5}$$

where $\theta_n(i) = -1 + \frac{2(i-1)}{n-1}$ generates n uniformly distributed values in $[-1, 1]$. In other words, $q_n(x)$ is a general quantizer that maps any real value in [-1,1] into one of n uniformly distributed values in $[-1, 1]$. For example, $q_2(x)$ is the sign function $\{\forall x \geq 0| \; sign(x) = 1, \;\; \forall x < 0| \; sign(x) = -1\}$ and $q_3(x)$ maps x into one of the three values $\{-1, 0, 1\}$.

To summarize, optimizing 10.4 identifies a mapping and a reconstruction function such that the quantized data in the space generated by the mapping function can be reconstructed accurately by the reconstruction function. It should be noted that an m-dimensional n-ary code is generated by $q_n(f(\mathbf{X}))$.

10.2.2.1 Linear Subspace Quantization (LSQ)

LSQ is a multidimensional n-ary coding method based on Subspace Reduction where linear functions are used as the mapping and the reconstruction functions in 10.4. Assume that $f(\mathbf{x}) = \mathbf{W}^\intercal \mathbf{x}$ and $\tilde{f}(\mathbf{x}) = \mathbf{V}^\intercal \mathbf{x}$ where $\mathbf{W} \in \mathbb{R}^{D \times m}$ and $\mathbf{V} \in \mathbb{R}^{m \times D}$. Employing the Frobenius norm as the regularizing function, the optimization problem in 10.4 becomes:

$$\min_{\mathbf{W},\mathbf{V}} \{\|\mathbf{X} - \mathbf{V}^\intercal q_n(\mathbf{W}^\intercal \mathbf{X})\|_F^2 + \lambda \|\mathbf{V}\|_F^2\} \tag{10.6}$$

To solve this problem, we propose a two step iterative algorithm. Subsequently, the convergence of the proposed algorithm will be proven.

• Learning LSQ:

The optimization for $\mathbf{W}$ and $\mathbf{V}$ in 10.6 can be solved by a two step iterative optimization algorithm (i.e. fixing one variable and updating the other). The steps are as follows:

1. ***Fix* $\mathbf{W}$ *and update* $\mathbf{V}$*:*** For a fixed $\mathbf{W}$, define $\mathbf{H} = q_n(\mathbf{W}^\mathsf{T}\mathbf{X})$; then we have a closed form solution for $\mathbf{V}$ as

$$\mathbf{V} = (\mathbf{H}\mathbf{H}^\mathsf{T} + \lambda\mathbf{I})^{-1}\mathbf{H}\mathbf{X}^\mathsf{T} \tag{10.7}$$

2. ***Fix* $\mathbf{V}$ *and update* $\mathbf{W}$*:*** In this step, $\mathbf{W}$ is updated as:

$$\mathbf{W} = \mathbf{V}^\dagger \tag{10.8}$$

   where $\mathbf{V}^\dagger$ is the Moore Penrose pseudoinverse of $\mathbf{V}$. In the following we prove that the pseudoinverse is an optimal solution for 10.6 when $\mathbf{V}$ is fixed.

The algorithm iterates between step 1 and 2 until there is no progress in minimizing 10.6.

• **Convergence of LSQ:**

In order to prove the convergence of the algorithm, we show that both steps reduce the objective value. The optimality of the first step can be easily shown by simple linear algebra. Here, we focus on proving that the second step reduces the objective value.

Given that $\underset{\mathbf{Y}}{\operatorname{argmin}}\{\|\mathbf{B} - \mathbf{A}\mathbf{Y}\|\} = \underset{\mathbf{Y}}{\operatorname{argmin}}\{\|\mathbf{A}^\dagger\mathbf{B} - \mathbf{Y}\|\}$, the solution of the optimization in 10.6 for fixed $\mathbf{V}$ is equivalent to the solution of the following problem:

$$\min_{\mathbf{W}}\{\|{\mathbf{V}^\dagger}^\mathsf{T}\mathbf{X} - q_n(\mathbf{W}^\mathsf{T}\mathbf{X})\|_F^2\} \tag{10.9}$$

Defining $\mathbf{M} = {\mathbf{V}^\dagger}^\mathsf{T}\mathbf{X}$, the optimal solution for $\mathbf{W}$ can be formulated as:

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}}\{\|\mathbf{M} - q_n(\mathbf{W}^\mathsf{T}\mathbf{X})\|_F^2\} \tag{10.10}$$

It should be noted that the optimal solution is not unique. Therefore, $\mathcal{W}$ is defined as the optimal solution set for $\mathbf{W}^*$. The goal is to prove that $\mathbf{V}^\dagger \in \mathcal{W}$.

Let $\mathbf{A}^* = {\mathbf{W}^*}^\mathsf{T}\mathbf{X}$. We first prove that $q_n(\mathbf{A}^*) = q_n(\mathbf{M})$. Suppose, to the contrary, that $q_n(\mathbf{A}^*) \neq q_n(\mathbf{M})$. Consequently, there should be at least one i and j, such that $q_n(A^*(i,j)) \neq q_n(M(i,j))$. Since $q_n(\mathbf{A}^*)$ is defined in the optimal solution of the optimization 10.10, its corresponding objective value should be less than that of any other feasible point. This leads to the conclusion that $(M(i,j) - q_n(A^*(i,j)))^2 < (M(i,j) - q_n(M(i,j)))^2$ (Note that even if more than one element differs between $q_n(\mathbf{A}^*)$ and $q_n(\mathbf{M})$, the inequality holds for at least one of them). However, this contradicts the definition of $q_n(x)$ in 10.5 since $q_n(x)$ should map

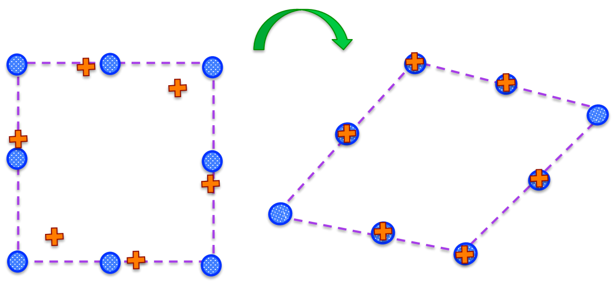


FIGURE 10.2: LSQ fitting 2D data using a q_3 quantizer. Data points are shown by orange crosses. The blue circles indicate the quantization levels in different dimensions. LSQ reduces the reconstruction error by performing a linear transformation over the quantized hyper-cube.

$M(i,j)$ into $q_n(A^*(i,j))$ (It should be noted that $q_n(A^*(i,j))$ is in the range of $q_n(M(i,j))$). So for any i and j, $q_n(A^*(i,j)) = q_n(M(i,j))$. Therefore, $q_n(\mathbf{M}) = q_n(\mathbf{A}^*)$. Considering the definition of $\mathbf{M}$ and $\mathbf{A}^*$ completes the proof that $\mathbf{V}^\dagger \in \mathcal{W}$.

Finally, since both steps in our optimization reduce the objective value, LSQ converges to a local optimal value of optimization 10.6.

• Relation to ITQ:

ITQ is a special case of LSQ when $n = 2$ and $\mathbf{W}$, $\mathbf{V}$ are rotation matrixes where $\mathbf{W}$=$\mathbf{V}^\mathsf{T}$. Our experiments show that the binary codes generated by LSQ leads to higher accuracies than the binary codes generated by ITQ.

• Geometrical Interpretation:

LSQ finds a linear transformation of a quantized hyper-cube that best fits the data. Figure 10.2 illustrates a simple 2D example in which the q_3 quantizer is fit to the data by a rotation.

10.2.2.2 LSQ as an n-ary Embedding

While binary encoding techniques try to minimize the reconstruction error, the resulting codes preserve similarities between samples. In other words, the Hamming distance in the binary space approximates the Euclidean distance in the original feature space. As a consequence of this property, these binary codes can be exploited as feature vectors for learning tasks in the embedded space. Many recent approaches based on this property have been proposed to make learning more efficient [190, 202].

In subspace clustering methods (e.g. CK-means), the cluster indices generated by the quantizer can not be viewed as a similarity preserving embedding. This is due to the fact that there are no constraints on assigning these indices to clusters. In subspace reduction methods (e.g. LSQ), each dimension of an n-ary code has a finite(discrete) set of real values as its domain. For each dimension, the distances between these discrete values correlates with the distances between the data points in the original feature space in the direction of that dimension. Therefore, the Euclidean distance in the quantized data correlates with the Euclidean distance in the original feature space.

One could post process CK-means to generate similarity(distance) preserving codes by assigning the appropriate indices to cluster centers after completion of the training stage. These indices can be obtained by finding a 1D subspace for each of the subspaces generated by CK-means. A simple model could compute PCA over the cluster centers in each subspace to reduce the cluster centers into 1D real values. However, in 10.4.5, a classification experiment is performed in which the n-ary codes are used as features. The result shows that, as an embedding, LSQ outperforms CK-Means by a large margin even after refining the CK-Means index assignments to clusters.

10.3 K-NN Retrieval using Data Encoding

A large source of computational cost in nearest neighbor search is the distance computation between the query and all the samples in the dataset. In order to speed up K-NN search, one can either speed up the computation of the distance function and/or reduce the number of distance computations by limiting the search space for a given query. We refer to the first strategy as *Distance Estimation* and the second as *Subset Indexing*. In the following subsections, we show how Subspace Clustering and Subspace Reduction coding techniques can be used for each of these strategies.

10.3.1 Retrieval by Distance Estimation

Data coding can reduce the cost of distance computation since the Euclidean distance can be efficiently estimated in the coding space.

• Distance estimation using Subspace Clustering n-ary codes:

Once data is coded, the Euclidean distance between two points can be estimated as the sum of distances between the assigned cluster centers to those data points in each subspace [194]. This is known as the *symmetric distance*. The distances between the n cluster centers in each subspace can be pre-computed in an $n \times n$ table. Then, computing the symmetric distance can be implemented efficiently by m look-ups and additions of table elements, one for each subspace.

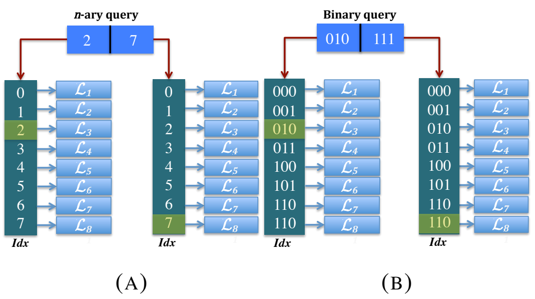

n-ary query
2
7
0 1 2 3 4 5 6 7
$\mathcal{L}_1$ $\mathcal{L}_2$ $\mathcal{L}_3$ $\mathcal{L}_4$ $\mathcal{L}_5$ $\mathcal{L}_6$ $\mathcal{L}_7$ $\mathcal{L}_8$
Idx
Binary query
010
111
000 001 010 011 100 101 110 110
Idx
(A)
(B)

FIGURE 10.3: Retrieval using Multiple Index Hashing (a) n-ary Coding: each dimension of the query is a key index to its corresponding table. Each table has n rows which points to set of samples in the base set with the same value in that dimension. (b) Binary Coding: In this case, binary codes are partitioned into sets of b consecutive bits (here $b = 3$) and each set is used as a key index for the corresponding table. Tables have 2^b rows containing samples with the same value in the corresponding bits.

More formally,

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{m} L_i(c(u_i(\mathbf{x})), c(u_i(\mathbf{y}))) \tag{10.11}$$

where $u_i(\mathbf{x})$ project $\mathbf{x}$ into the i^{th} subspace, $c(u)$ is the cluster index to which u belongs, and L_i is the pre-computed distance table for the i^{th} subspace. If we consider $\mathbf{x}$ as the query and $\mathbf{y}$ as a data point from the database, $c(u_i(\mathbf{y})))$ can be pre-computed. Therefore the complexity is $\mathcal{O}(mN)$ for each query, where N is the total number of points in the database.

• Distance estimation using subspace reduction n-ary codes:

As mentioned earlier, the Euclidean distance between quantized data by subspace reduction approximates the Euclidean distance in the original feature space. Therefore we need only compute the distance between coded values. This has complexity $\mathcal{O}(mN)$, which is the same as the complexity of subspace clustering.

• Distance estimation using binary codes:

For the binary codes, Hamming distance is used as the distance metric. Computing Hamming distances using m-bit binary codes has complexity $\mathcal{O}(mN)$ for each query.

10.3.2 Retrieval by Subset Indexing

Another way to speed up nearest neighbor search is to limit the search space. Hashing techniques [38] and tree based methods[176] limit the search space by constraining search to a subset of samples in the database. This is accomplished by indexing the data into a data structure (e.g.

hash tables or search tree) at training time. Multiple index hashing [180, 203] is one such data structure that can be used for binary and n-ary codes.

• **Multiple index hashing using n-ary codes:**

In this approach, for m-dimensional n-ary coding, we create an index table $\mathcal{T}_i,\ i = 1, \ldots, m$ for each dimension. Each table has n tuples $(Idx_j, \mathcal{L}_j),\ j = 1, \ldots, n$. where Idx_j corresponds to one of the n values in a dimension of the code and $\mathcal{L}_j$ is a list of those data points' indices such that the value of the i^{th} dimension in their code is Idx_j. At query time, for each dimension of the code, a set of data indices is retrieved. Figure 10.3(a) illustrates this technique. For each index in the union of these sets, we assign a score s between 1 and m which indicates that a particular index has been retrieved from s dimensions. The samples with higher scores are more likely to be similar to the query sample. By sorting the indices based on their score, we can choose the top-K samples as the K-NN's. If the total number of retrieved indices were less than K, we change the value in one of dimensions in the query code that has minimum distance to the quantized query point in the original space. Then we retrieve a new set and repeat the process until the total size of the retrieval set is greater than or equal to K.

• **Multiple index hashing with binary codes:**

Similar to n-ary codes, binary codes can be used for multiple index hashing. However, in this case each set of b consecutive bits are grouped together to create the indices for accessing the tables. Considering $b = log(n)$, this partitioned binary code can be seen as an n-ary code. As a result, the same technique can be applied for multiple index hashing as discussed previously. This case can be seen in Figure 10.3(b).

10.3.2.1 Subset Indexing: Binary or n-ary Coding?

As mentioned earlier, n-ary coding does not always outperforms binary coding for large-scale retrieval. More precisely, when Subset Indexing is used to reduce the search cost, binary coding achieves better search accuracy. This is due to the fact that quantization does not necessarily preserve the similarities (or distances) between data. In other words, a good quantizer Q that minimizes the quantization error in 10.1, does not always preserve relative distances between data. Formally:

$$\begin{aligned} &\|\mathbf{x}_1 - \mathbf{x}_2\| \leq \|\mathbf{x}_1 - \mathbf{x}_3\| \nRightarrow \\ &\|Q(\mathbf{x}_1) - Q(\mathbf{x}_2)\| \leq \|Q(\mathbf{x}_1) - Q(\mathbf{x}_3)\| \end{aligned} \tag{10.12}$$

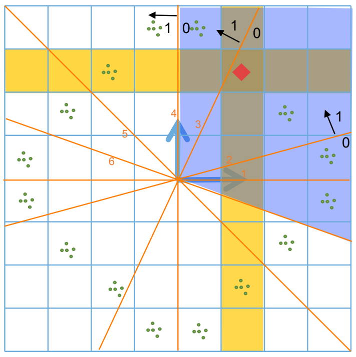


FIGURE 10.4: Binary vs. n-ary for ANN

This is important when retrieval is carried out by subset indexing. There, binary codes may retrieve the nearest neighbors better than n-ary codes. Figure 10.4 illustrate an example of 2-dimensional 8-ary codes and their corresponding binary codes, which have 6 bits ($6 = 2\log(8)$). Each bit is generated by a line based on which side of the line the point lies. The green dots are the points in the database and the red diamond is a query point. In this figure the binary code for the query sample is 110000. In the subspace clustering view, we cluster each dimension into 8 clusters. In this case, all points in the yellow region will be retrieved by multiple index hashing. As can be seen, none of the actual nearest neighbors can be retrieved. But, when we use the binary codes for multiple index hashing all the actual nearest neighbors are retrieved. This is the blue region (i.e. the union of the region created by the first three bits (110) and the second three bits (000) of the query code). Our experimental evaluation confirms that when subset indexing is used for retrieval, binary codes outperform n-ary codes. Although, n-ary codes are more accurate for quantization, they are not accurate for ANN with subset indexing.

10.4 Experiments

We report experiments on three well-known datasets, namely *GIST1M* [194], *CIFAR10* [204], and a subset of *ImageNet* [193] which is used for the ILSVRC2012 challenge. GIST1M contains 1M base feature vectors, 500K training samples and 1K query samples. For CIFAR10, we randomly selected 20K samples as our training set, 500 samples as query images and the remaining 39500 images as the base samples. Raw pixel values are used as features for this dataset. The ImageNet ILSVRC2012 dataset consists of 500K training samples, 250K base samples, and 1K query images. We used ConvNet as the state-of-the-art feature extractor for this dataset. The ConvNet features are extracted by Caffe [205].

Following [200], we used recall as the performance measure for retrieval. The training set is used to train the coding model and the learned model is applied for coding the base and query

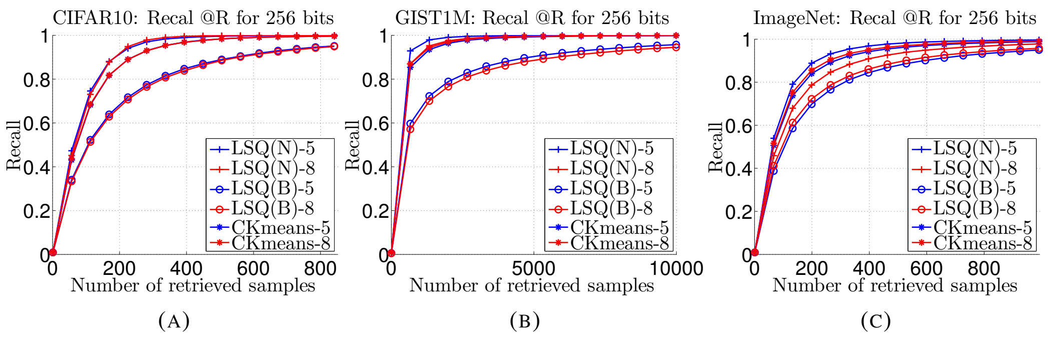


FIGURE 10.5: The Recall@R curves for retrieval using distance estimation for 256 bits. Each diagram shows the curve for different methods and different numbers of bits per code dimension. (a) Results on CIFAR10 dataset. (b) Results on GIST1M dataset. (c) Results on ImageNet dataset.

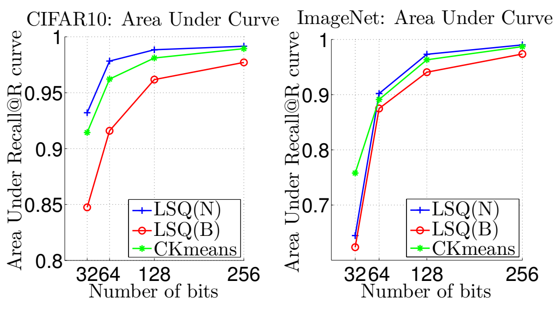
set. For each point in the query set, we find its R nearest neighbors and report the recall at R. By varying R we draw the recall curves.

As mentioned earlier, retrieval can be made faster using two approaches: distance estimation and subset indexing. The performance of different methods can vary with respect to which approach is used. Therefore, each method is examined with respect to both and an analysis is presented. The nearest neighbors in the original feature space are defined as ground truth for each query image. For making the comparison fair, in each experiment the number of bits which can be used by each coding method is limited to the same fixed budget. *e.g.* a 2 dimensional 8-ary code requires 6 bits of memory. (3 bits per code dimension).

10.4.1 Retrieval using Distance Estimation

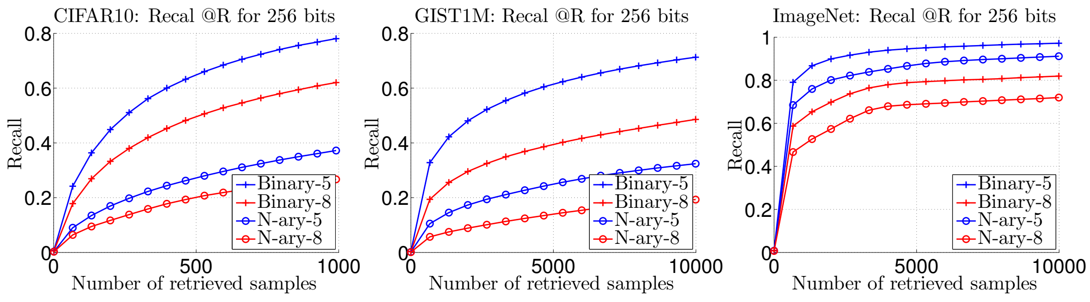
Figure 10.5, shows the Recall@R curves on different datasets using a budget of 256 bits. In this figure, LSQ(N) and LSQ(B) refer to the n-ary and binary versions of the LSQ method respectively. The recall@R curves are shown for different number of bits per code dimension, which controls the number of quantization steps for n-ary encoders (e.g. for LSQ(N)-5 or LSQ(B)-5 the quantizer has 32 levels or 5 bits). As can be seen, the performance of n-ary codes is better than binary codes. Also, LSQ outperforms CK-means on all three datasets.

Figure 10.6 explores the effect of the number of bits on the different methods. We fixed the number of bits per code dimension to 5 (e.g. the CK-means algorithm would learn 32 clusters per segment) and report the area under the Recall@R curve. Again, LSQ performs better than CK-means.



(A) (B)

FIGURE 10.6: The Area Under Recall@R curves for retrieval using 5 bits per code dimension (i.e. 32 quantization steps). Each diagram shows the curve for different methods and different amount of total bit budget. (a) Results on CIFAR10 dataset. (b) Results on ImageNet dataset.



(A) (B) (C)

FIGURE 10.7: The Recall@R curves for retrieval using subset indexing for 256 bits. Each diagram shows the curve for the best n-ary and binary coding method in this task (CKmeans and Binary LSQ respectively). (a) Results on CIFAR10 dataset. (b) Results on GIST1M dataset. (c) Results on ImageNet dataset.

10.4.2 Retrieval using Subset Indexing

As discussed in section 10.3.2, either binary or n-ary coding can be used to speed up search with Subset Indexing. This approach limits the search to a small number of samples by indexing subsets of the database (subset indexing). Here, the performance of binary and n-ary coding is compared. We compare the retrieval results of the best n-ary encoding for this task(CK-means) to the best binary coding(the binary version of LSQ).

Figure 10.7, shows the recall@R curves for this experiment with varying numbers of bits per code dimension for a fixed budget of 256 bits. In *N-ary-k*, k bits are used for quantizing each dimension and additionally indexing in the multi-index hashing method(i.e. 2^k quantization steps for each dimension). Similarly, in *Binary-k*, k consecutive bits are used for indexing in the hashing method. The effect of changing the budget of the encoder on the retrieval task can be seen in Figure 10.8. These figures illustrate that the binary encoding techniques outperform the n-ary encoders when the subset indexing technique is used, as discussed in Section 10.3.2.

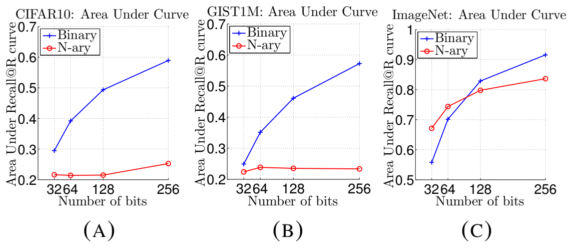


FIGURE 10.8: The Area Under Recall@R curves for retrieval using 5 bits per code dimension. Each diagram shows the curve for the best n-ary and binary coding method for this task and different bit budgets. (a) Results on CIFAR10 dataset. (b) Results on GIST1M dataset. (c) Results on ImageNet dataset.

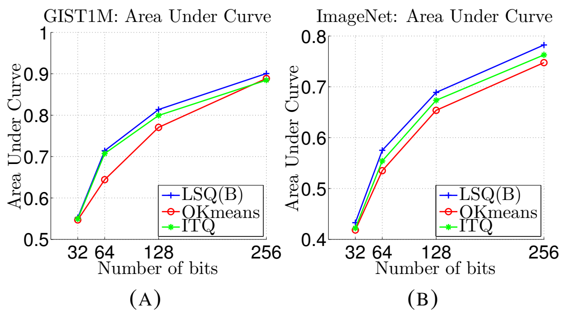


FIGURE 10.9: The Area Under precision recall curves for binary methods. Each diagram shows the curve for different methods and different bit budgets. (a) Results on GIST1M dataset. (b) Results on ImageNet dataset.

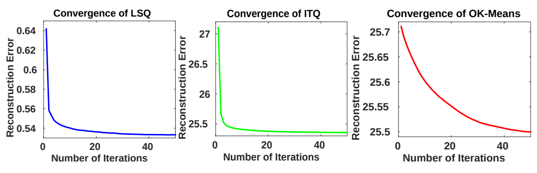
Discussion: These experiments confirm that when retrieval is performed by **distance estimation**, it is better to use n-ary coding with $n > 2$ based on subspace reduction (e.g. LSQ). On the other hand, when **subset indexing** is used for retrieval, binary coding outperforms n-ary coding.

10.4.3 Comparison of binary coding methods

Both CK-means and LSQ can be viewed as generalizations of binary encoding where the number of quantization steps can be more than two. Here, the number of quantization steps is set to two and the binary versions of LSQ and CK-means (namely LSQ(B) and OK-means respectively) are compared with ITQ using subset indexing. Figure 10.9, shows the area under the recall precision curve for these three binary coding methods under varying bit budgets. As can be seen, the binary version of LSQ outperforms ITQ and the binary version of CK-means.

10.4.4 Convergence of the algorithms

In Figure 10.10, the convergence of different binary coding methods are shown. For this experiment, GIST1M is used. As can be seen, LSQ converges much faster than OK-Means. Also note that the final reconstruction error of LSQ is much smaller than ITQ and OK-means, reflecting the fact that LSQ reconstructs the data more accurately using the same memory budget.



(A) (B) (C)

FIGURE 10.10: The convergence of different coding algorithms. (a) LSQ(B) (b) ITQ (c) OK-Means. Note different scale of reconstruction error are required.

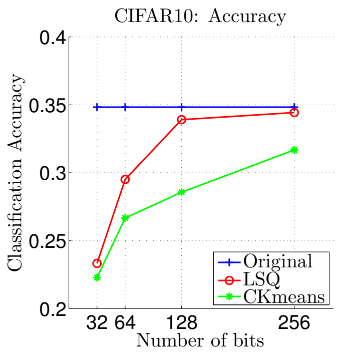


FIGURE 10.11: Classification accuracy using the codes of different methods as feature vectors.

10.4.5 n-ary Codes as Feature Vectors

The codes constructed by LSQ can be used as feature vectors to perform learning tasks. Figure 10.11 shows the performance evaluation of a classification task using different codings as features. As proposed in sec 10.2.2.2, for CK-means, we refine the index assignments to clusters by mapping the cluster centers in each subspace into a one dimensional space using PCA and convert each dimension of the code to the corresponding value in this 1D space. It can be seen that our proposed quantization method outperforms CK-means even after refining the CK-Means index assignments.

Chapter 11

Conclusion

We showed how an efficient and rich representation of visual data can be obtained via several learning process. We also showed how we can use these data for better understanding of visual information when dealing with large-scale databases. Over the past decade, the usage of large databases for object and speech recognition has shown that simple learning methods, such as linear classifiers and K-Nearest Neighbors, can achieve comparable performance to complex learning methods. This has led to widespread discussion about how much *data matters*, which posits that having more data is more effective than having a complex model. Resolving the extent to which *data matters* is confounded by computational and algorithmic challenges in learning a complex model on a large-scale dataset.

On the one hand, the problem of learning complex models on big data has been addressed by investigating the use of high-performance and parallel computing (for example, deep neural networks). A complementary approach to dealing with big data is developing efficient data representations which enable efficient processing. Representing data using compact binary codes is one of the most powerful ways to achieve efficient data representation. It has received significant attention during the past years. In my research, I contributed to progress on constructing effective binary representations of visual data by shifting the goal of image retrieval using binary codes from near-duplicate retrieval to semantic retrieval. My goal was to find efficient representations of data that A) capture enough semantics to enable accurate recognition and understanding in a variety of tasks, and B) can be extracted efficiently from the original feature space (in my research mostly visual and textual space) to make it practical to be applied to large-scale data and complex models. Finding such a representation leads to significant progress in a wide range of machine learning problems. In pursuing these goals, I have completed several projects on image and text understanding. Some of them are published in prestigious conferences and others are submitted or still in progress. I showed how a certain class of binary codes can efficiently and accurately solve a variety of problems in computer vision. I introduced the notion of predictability of code bits, which enables each bit of the binary code to be reliably predicted from the original feature space. I showed that predictability plays an important role in recognition problems. In recent work, we showed that domain adaptation can significantly benefit from predictable binary codes. I elucidated a close relation between predictable code bits and visual attributes (category independent semantic descriptors which form the basis for most zero shot - no training data - visual learning algorithms). My research has shown how these predictable codes can be effectively applied to a wide variety of tasks (patch based image restoration, cross modality hashing, video clustering, biometric recognition, etc.). In the remainder of this statement I will describe what I have accomplished so far and the main problems that I plan to address in the future.

Bibliography

[1] P. Jain, S. Vijayanarasimhan, and K. Grauman. Hashing Hyperplane Queries to Near Points with Applications to Large-Scale Active Learning. In *NIPS*, 2010.

[2] Mohammad Rastegari, Chen Fang, and Lorenzo Torresani. Scalable object-class retrieval with approximate and top-k ranking. In *ICCV*, 2011.

[3] Mohammad Rastegari, Ali Farhadi, and David A. Forsyth. Attribute discovery via predictable discriminative binary codes. In *ECCV (6)*, 2012.

[4] Jonghyun Choi, Mohammad Rastegari, Ali Farhadi, and Larry S. Davis. Adding unlabeled samples to categories by learned attributes. In *IEEE Conference Computer Vision and Pattern Recognition (CVPR)*, 2013.

[5] Fatemeh Mirrashed and Mohammad Rastegari. Domain adaptive classification. In *International Conference on Computer Vision (ICCV)*, 2013.

[6] Mohammad Rastegari, Jonghyun Choi, Shobeir Fakhraei, Hal Daume, and Larry Davis. Dual-view predictable hashing. In *International Conference on Machine Learning (ICML)*, 2013.

[7] Mohammad Rastegari, Ali Diba, Devi Parikh, and Ali Farhadi. Multi-attribute querry: To merge or not to merge? In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

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

[9] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. A discriminative framework for texture and object recognition using local image features. In *Toward Category-Level Object Recognition*, pages 423–442, 2006.

[10] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *CVPR (1)*, pages 886–893, 2005.

[11] P.V. Gehler and S. Nowozin. On feature combination for multiclass object classification. In *ICCV*, 2009.

[12] Hervé Jegou, Matthijs Douze, and Cordelia Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *European Conf. Computer Vision*, October 2008.

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

[14] O. Chum, J. Philbin, and A. Zisserman. Near duplicate image detection: min-hash and tf-idf weighting. In *Proceedings of the British Machine Vision Conference*, 2008.

[15] David Nistér and Henrik Stewénius. Scalable recognition with a vocabulary tree. In *CVPR (2)*, pages 2161–2168, 2006.

[16] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *CVPR*, 2008.

[17] Antonio Torralba, Robert Fergus, and Yair Weiss. Small codes and large image databases for recognition. In *CVPR*, 2008.

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

[19] A. Farhadi, I. Endres, D. Hoiem, and D.A. Forsyth. Describing objects by their attributes. In *CVPR*, 2009.

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

[21] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar. Attribute and Simile Classifiers for Face Verification. In *IEEE International Conference on Computer Vision (ICCV)*, Oct 2009.

[22] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[23] Lorenzo Torresani, Martin Szummer, and Andrew W. Fitzgibbon. Learning query-dependent prefilters for scalable image retrieval. In *CVPR*, pages 2615–2622, 2009.

[24] Eric P. Xing Li-Jia Li, Hao Su and Li Fei-Fei. Object bank: A high-level image representation for scene classification semantic feature sparsification. In *Neural Information Processing Systems (NIPS)*, Vancouver, Canada, December 2010.

[25] Kristen Grauman and Trevor Darrell. Pyramid match hashing: Sub-linear time indexing over partial correspondences. In *CVPR*, 2007.

[26] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.

[27] G. Shakhnarovich, T. Darrell, P. Indyk, and editors. *Nearest-Neighbors methods in Learning and Vision: Theory and Practice*. Springer Series in Statistics. MIT Press., New York, NY, USA, 2006.

[28] Prateek Jain, Brian Kulis, and Kristen Grauman. Fast image search for learned metrics. In *CVPR*, 2008.

[29] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, pages 2130–2137, 2009.

[30] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[31] Mingkui Tan, Li Wang, and Ivor W. Tsang. Learning sparse svm for feature selection on very high dimensional datasets. In *ICML*, pages 1047–1054, 2010.

[32] Herve Jegou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, pages 3304–3311, 2010.

[33] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 33(1): 117–128, jan 2011. URL http://lear.inrialpes.fr/pubs/2011/JDS11. to appear.

[34] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[35] A. Berg, J. Deng, and L. Fei-Fei. Large scale visual recognition challenge, 2010. http://www.image-net.org/challenges/LSVRC/2010/.

[36] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li 0002. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[37] Peter Lyman, Hal R. Varian, Peter Charles, Nathan Good, Laheem L. Jordan, and Joyojeet Pal. How much information?, 2003.

[38] P. Indyk A. Gionis and R. Motwani. Similarity search in high dimensions via hashing. *VLDB*, 1999.

[39] Gregory Shakhnarovich, Paul A. Viola, and Trevor Darrell. Fast pose estimation with parameter-sensitive hashing. In *ICCV*, 2003.

[40] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2009.

[41] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. 2009.

[42] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 2009.

[43] A. Torralba, R Fergus, , and Y. Weiss. Small codes and large image databases for recognition. In *CVPR*, 2008.

[44] Ruslan Salakhutdinov and Geoffrey Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *AISTATS*, 2007.

[45] Mohammad Norouzi and David Fleet. Minimal loss hashing for compact binary codes. In *ICML*, 2011.

[46] Yair Weiss, Antonio B. Torralba, and Robert Fergus. Spectral hashing. In *NIPS*, 2008.

[47] M. Raginsky and S. Lazebnik. Locality sensitive binary codes from shift-invariant kernels. In *NIPS*, 2009.

[48] Ali Rahimi and Ben Recht. Random features for large-scale kernel machines. In *NIPS*, 2007.

[49] Ruei sung Lin, David A. Ross, and Jay Yagnik. Spec hashing: Similarity preserving algorithm for entropy-based coding. In *CVPR*, 2010.

[50] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, 2010.

[51] Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011.

[52] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-Supervised Hashing for Scalable Image Retrieval. In *CVPR*, 2010.

[53] Ali Farhadi and David Forsyth. Transfer learning in sign language. In *CVPR*, 2007.

[54] Ali Farhadi and Mostafa Kamali Tabrizi. Learning to recognize activities from the wrong view point. In *ECCV*, 2008.

[55] Lorenzo Torresani, Martin Szummer, and Andrew Fitzgibbon. Efficient object category recognition using classemes. In *ECCV*, 2010.

[56] Ali Farhadi, Ian Endres, Derek Hoiem, and David Forsyth. Describing objects by their attributes. In *CVPR*, 2009.

[57] Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by betweenclass attribute transfer. In *CVPR*, 2009.

[58] Devi Parikh and Kristen Grauman. Interactively building a discriminative vocabulary of nameable attributes. In *CVPR*, pages 1681–1688, 2011.

[59] Kun Duan, Devi Parikh, David Crandall, and Kristen Grauman. Discovering localized attributes for fine-grained recognition. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2012.

[60] Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Constrained semi-supervised learning using attributes and comparative attributes. In *ECCV*, 2012.

[61] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Sequential projection learning for hashing with compact codes. In *ICML*, 2010.

[62] G. Griffin, A. Holub, and P. Perona. The Caltech-256. Technical report, 2007.

[63] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

[64] G. J. Burghouts and J. M. Geusebroek. Performance evaluation of local colour invariants. *CVIU*, 2009.

[65] Peter V. Gehler and Sebastian Nowozin. On feature combination for multiclass object classification. In *ICCV*, 2009.

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

[67] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 2008.

[68] Alessandro Bergamo and Lorenzo Torresani. Picodes: Learning a compact code fornovel-category recognition. In *NIPS*, 2011.

[69] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1), 2011.

[70] Aristides Gionis, Piotr Indyk, Rajeev Motwani, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[71] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.

[72] W. Liu, J. Wang, R. Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081, 2012.

[73] D. R. Hardoon, S. Szedmak, O. Szedmak, and J. Shawe-taylor. Canonical correlation analysis; An overview with application to learning methods. Technical report, University of London, 2003.

[74] S. J. Hwang and K. Grauman. Accounting for the Relative Importance of Objects in Image Retrieval. In *BMVC*, 2010.

[75] S. J. Hwang and K. Grauman. Learning the Relative Importance of Objects from Tagged Images for Retrieval and Cross-Modal Search. *IJCV*, 100(2):134–153, 2012.

[76] Ali Farhadi, Mohsen Hejrati, Mohammad Amin Sadeghi, Peter Young, Cyrus Rashtchian, Julia Hockenmaier, and David Forsyth. Every picture tells a story: generating sentences from images. In *ECCV*, pages 15–29, Berlin, Heidelberg, 2010.

[77] Cyrus Rashtchian, Peter Young, Micah Hodosh, and Julia Hockenmaier. Collecting image annotations using amazon's mechanical turk. In *CSLDAMT*, pages 139–147, 2010.

[78] G. Kulkarni, V. Premraj, S. Dhar, Siming Li, Yejin Choi, A.C. Berg, and T.L. Berg. Baby talk: Understanding and generating simple image descriptions. In *CVPR*, pages 1601 –1608, june 2011.

[79] Siming Li, Girish Kulkarni, Tamara L. Berg, Alexander C. Berg, and Yejin Choi. Composing simple image descriptions using web-scale n-grams. In *CoNLL*, pages 220–228, 2011.

[80] J. Masci, M. M. Bronstein, A. A. Bronstein, and Jürgen Schmidhuber. Multimodal similarity-preserving hashing. *CoRR*, abs/1207.1522, 2012.

[81] Y. Zhen and Dit-Yan Yeung. Co-Regularized Hashing for Multimodal Data. In *NIPS*, 2012.

[82] Shaishav Kumar and Raghavendra Udupa. Learning Hash Functions for Cross-View Similarity Search. In *IJCAI*, 2011.

[83] Vicente Ordonez, Girish Kulkarni, and Tamara L. Berg. Im2text: Describing images using 1 million captioned photographs. In *NIPS*, pages 1143–1151, 2011.

[84] Jesse Dodge, Amit Goyal, Xufeng Han, Alyssa Mensch, Margaret Mitchell, Karl Stratos, Kota Yamaguchi, Yejin Choi, Hal Daumé III, Alexander C. Berg, and Tamara L. Berg. Detecting visual text. In *HLT-NAACL*, pages 762–772, 2012.

[85] Polina Kuznetsova, Vicente Ordonez, Alexander C. Berg, Tamara L. Berg, and Yejin Choi. Collective generation of natural image descriptions. In *ACL (1)*, pages 359–368, 2012.

[86] Alexander C. Berg, Tamara L. Berg, Hal Daumé III, Jesse Dodge, Amit Goyal, Xufeng Han, Alyssa Mensch, Margaret Mitchell, Aneesh Sood, Karl Stratos, and Kota Yamaguchi. Understanding and predicting importance in images. In *CVPR*, pages 3562–3569, 2012.

[87] Y. Gong, Q. Ke, M. Isard, and S. Lazebnik. A Multi-View Embedding Space for Modeling Internet Images, Tags, and their Semantics. *CoRR*, abs/1212.4522, 2012.

[88] G. Patterson and J. Hays. SUN Attribute Database: Discovering, Annotating, and Recognizing Scene Attributes. In *CVPR*, 2012.

[89] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *IJCV*, 2001.

[90] D. Lin. An Information-Theoretic Definition of Similarity. In *ICML*, pages 296–304, 1998.

[91] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. SUN Database: Large-scale Scene Recognition from Abbey to Zoo. In *CVPR*, 2010.

[92] R. E. Fan, K. W. Chang, C. J. Hsieh, X. R. Wang, and C. J. Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 2008.

[93] W. Zhang, S. X. Yu, and Shang-Hua Teng. PowerSVM: Generalization with Exemplar Classification Uncertainty. In *CVPR*, 2012.

[94] T. L. Berg, A. Sorokin, G. Wang, D. A. Forsyth, D. Hoiem, A. Farhadi, and I. Endres. It's All About the Data. In *Proceedings of the IEEE, Special Issue on Internet Vision*, 2010.

[95] B. Settles. Active Learning Literature Survey. Technical report, 2009.

[96] A. Parkash and D. Parikh. Attributes for Classifier Feedback. In *ECCV*, 2012.

[97] X. Zhu. Semi-Supervised Learning Literature Survey. Technical report, 2008.

[98] R. Fergus, Y. Weiss, and A. Torralba. Semi-supervised Learning in Gigantic Image Collections. In *NIPS*, 2009.

[99] A. Shrivastava, S. Singh, and A. Gupta. Constrained semi-supervised learning using attributes and comparative attributes. In *ECCV*, 2012.

[100] J. J. Lim, R. Salakhutdinov, and A. Torralba. Transfer Learning by Borrowing Examples for Multiclass Object Detection. In *NIPS*, 2011.

[101] J. Kim and K. Grauman. Shape Sharing for Object Segmentation. In *ECCV*, 2012.

[102] S. J. Pan and Q. Yang. A Survey on Transfer Learning. *IEEE Trans. on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[103] R. Salakhutdinov, A. Torralba, and J. Tenenbaum. Learning to Share Visual Appearance for Multiclass Object Detection. In *CVPR*, 2011.

[104] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth. Describing Objects by their Attributes. In *CVPR*, 2009.

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

[106] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar. Attribute and Simile Classifiers for Face Verification. In *ICCV*, 2009.

[107] M. Rastegari, A. Farhadi, and D. Forsyth. Attribute Discovery via Predictable Discriminative Binary Codes. In *ECCV*, 2012.

[108] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

[109] L. Torresani, M. Szummer, and A. Fitzgibbon. Efficient Object Category Recognition Using Classmes. In *ECCV*, 2010.

[110] T. Malisiewicz, A. Gupta, and A. A. Efros. Ensemble of Exemplar-SVMs for Object Detection and Beyond. In *ICCV*, 2011.

[111] A. Vedaldi and A. Zisserman. Efficient Additive Kernels via Explicit Feature Maps. *IEEE Trans. PAMI*, 2011.

[112] J. Smith, M. Naphade, and A. Natsev. Multimedia semantic indexing using model vectors. In *ICME*, 2003.

[113] N. Rasiwasia, P. Moreno, and N. Vasconcelos. Bridging the gap: Query by semantic example. *Trans Multimedia*, 9(5), Aug 2007.

[114] M. Naphade, J. Smith, J. Tesic, S. Chang, W. Hsu, L. Kennedy, A. Hauptmann, and J. Curtis. Large-scale concept ontology for multimedia. *IEEE Multimedia*, 13(3), 2006.

[115] E. Zavesky and S.-F. Chang. Cuzero: Embracing the frontier of interactive visual search for informed users. In *ACM MIR*, 2008.

[116] Matthijs Douze, Arnau Ramisa, and Cordelia Schmid. Combining attributes and fisher vectors for efficient image retrieval. In *CVPR*, 2011.

[117] Xiaogang Wang, Ke Liu, and Xiaoou Tang. Query-specific visual semantic spaces for web image re-ranking. In *CVPR*, 2011.

[118] B. Saleh, A. Farhadi, and A. Elgammal. Object-centric anomaly detection by atribute-based reasoning. In *CVPR*, 2013.

[119] B. Siddiquie, R. S. Feris, and L. S. Davis. Image Ranking and Retrieval based on Multi-Attribute Queries. In *CVPR*, 2011.

[120] Walter Scheirer, Neeraj Kumar, Peter N. Belhumeur, and Terrance E. Boult. Multi-attribute spaces: Calibration for attribute fusion and similarity search. In *The 25th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2012.

[121] M. A. Sadeghi and A. Farhadi. Recognition Using Visual Phrases. In *CVPR*, 2011.

[122] Congcong Li, Devi Parikh, and Tsuhan Chen. Automatic discovery of groups of objects for scene understanding. In *CVPR*, 2012.

[123] Yair Weiss, Rob Fergus, and Antonio Torralba. Multidimensional spectral hashing. In *ECCV (5)*, 2012.

[124] Mohammad Rastegari, Ali Farhadi, and David A. Forsyth. Attribute discovery via predictable discriminative binary codes. In *ECCV (6)*, 2012.

[125] A. Wagner, J. Wright, A. Ganesh, Z. Zhou, H. Mobahi, and Y. Ma. Towards a Practical Face Recognition System: Robust Alignment and Illumination by Sparse Representation. *IEEE PAMI*, 2011.

[126] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical report, California Institute of Technology, 2010.

[127] Kun Duan, Devi Parikh, David J. Crandall, and Kristen Grauman. Discovering localized attributes for fine-grained recognition. In *CVPR*, 2012.

[128] M. Pawan Kumar, B. Packer, and D. Koller. Self-Paced Learning for Latent Variable Models. In *NIPS*, 2010.

[129] K. Saenko, B. Kulis, M. Fritz, and T. Darrell. Adapting visual category models to new domains. In *ECCV*, 2010.

[130] A. Bergamo and L. Torresani. Exploiting weakly-labeled web images to improve object classification: A domain adaptation approach. In *NIPS*, 2010.

[131] Hal Daume III, Abhishek Kumar, and Avishek Saha. Co-regularization based semi-supervised domain adaptation. In *NIPS*, 2010.

[132] Hal Daume III. Frustratingly easy domain adaptation. 2007.

[133] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Conference on Empirical Methods in Natural Language Processing*, 2006.

[134] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL*, 2007.

[135] Boqing Gong, Yuan Shi, Kristen Grauman, and Fei Sha. Connecting the dots with landmarks: Discriminatively learning domain-invariant features for unsupervised domain adaptation. In *ICML*, 2013.

[136] Sinno Jialin Pan, Ivor W. Tsang, James T. Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2), 2011.

[137] R. Gopalan ad R. Li and R. Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *ICCV*, 2011.

[138] Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *CVPR*, 2012.

[139] Mohammad Rastegari, Ali Farhadi, and David Forsyth. Attribute discovery via predictable discriminative binary codes. In *ECCV*, 2012.

[140] A. Torralba and A. Efros. Unbiased look at dataset bias. In *CVPR*, 2011.

[141] Aditya Khosla, Tinghui Zhou, Tomasz Malisiewicz, Alexei Efros, and Antonio Torralba. Undoing the damage of dataset bias. In *ECCV*, 2012.

[142] Hal Daumé, III and Daniel Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*, 2006.

[143] Ali Farhadi, David A. Forsyth, and Ryan White. Transfer learning in sign language. In *CVPR*, 2007.

[144] Ali Farhadi and Mostafa Kamali Tabrizi. Learning to recognize activities from the wrong view point. In *ECCV*, 2008.

[145] B. Kulis, K. Saenko, and T. Darrell. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *CVPR*, 2011.

[146] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007. URL http://authors.library.caltech.edu/7694.

[147] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *In ECCV*, 2006.

[148] Huang J., A.J. Gretton, K.M. Borgwardt, and B. Scholkopf. Correcting sample selection bias by unlabeled data. In *NIPS*, 2007.

[149] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88, 2010.

[150] Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010.

[151] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Labelme: A database and web-based tool for image annotation. *IJCV*, 2007.

[152] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *IJCV*, 2(60): 91–110, 2004.

[153] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. "
urlhttp://www.vlfeat.org/", 2008.

[154] Mohammad Norouzi, Ali Punjani, and David J Fleet. Fast search in hamming space with multi-index hashing. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3108–3115. IEEE, 2012.

[155] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[156] Y. Weiss, A. Torralba, and R. Fergus. Spectral Hashing. In *NIPS*, 2008.

[157] Yair Weiss, Rob Fergus, and Antonio Torralba. Multidimensional spectral hashing. In *Proceedings of the 12th European conference on Computer Vision (ECCV)*, pages 340–353. Springer-Verlag, 2012.

[158] Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 817–824. IEEE, 2011.

[159] Mohammd Norouzi and David Fleet. Cartesian k-means. In *Computer Vision and Pattern Recognition*. IEEE, 2013.

[160] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Searching with quantization: approximate nearest neighbor search using short codes and distance estimators. Technical Report RR-7020, INRIA, aug 2009.

[161] M. Norouzi and D. J. Fleet. Minimal Loss Hashing for Compact Binary Codes. In *ICML*, 2011.

[162] W Liu, J Wang, R Ji, YG Jiang, and SF Chang. Supervised hashing with kernels. *Proceedings of Computer Vision and Pattern Recognition*, 2012.

[163] Zhongming Jin, Yao Hu, Yue Lin, Debing Zhang, Shiding Lin, Deng Cai, and Xuelong Li. Complementary projection hashing. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2013.

[164] Guosheng Lin, Chunhua Shen, David Suter, and Anton van den Hengel. A general two-step approach to learning-based hashing. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2013.

[165] Fumin Shen, Chunhua Shen, Qinfeng Shi, Anton van den Hengel, and Zhenmin Tang. Inductive hashing on manifolds. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[166] Xianglong Liu, Junfeng He, Bo Lang, and Shih-Fu Chang. Hash bit selection: A unified solution for selection problems in hashing. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[167] Yue Lin, Rong Jin, Deng Cai, Shuicheng Yan, and Xuelong Li. Compressed hashing. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[168] Kaiming He, Fang Wen, and Jian Sun. K-means hashing: An affinity-preserving quantization method for learning binary compact codes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[169] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization for approximate nearest neighbor search. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[170] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[171] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3485–3492. IEEE, 2010.

[172] Genevieve Patterson and James Hays. Sun attribute database: Discovering, annotating, and recognizing scene attributes. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2751–2758. IEEE, 2012.

[173] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Proceedings of the international conference on very large data bases*, pages 518–529, 1999.

[174] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *The Neural Information Processing Systems*, 2009.

[175] Bernhard Scholkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *Advances in kernel methods-support vector learning*. Citeseer, 1999.

[176] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1986.

[177] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36, 2014.

[178] Marius Muja and David G. Lowe. Fast matching of binary features. In *Computer and Robot Vision (CRV)*, pages 404–410, 2012.

[179] Liang Zheng, Shengjin Wang, and Qi Tian. Coupled binary embedding for large-scale image retrieval. *Image Processing, IEEE Transactions on*, 2014.

[180] Mohammad Norouzi and Ali Pournaji. Fast search in hamming space with multi-index hashing. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, 2012.

[181] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *Pattern Analysis and Machine Intellingence*, 34(3), 2011.

[182] Subhransu Maji, Alexander C. Berg, and Jitendra Malik. Efficient classification for additive kernel svms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2013.

[183] Yunchao Gong, Sanjiv Kumar, Henry A Rowley, and Svetlana Lazebnik. Learning binary codes for high-dimensional data using bilinear projections. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 484–491. IEEE, 2013.

[184] Felix X Yu, Sanjiv Kumar, Yunchao Gong, and Shih-Fu Chang. Circulant binary embedding. *arXiv preprint arXiv:1405.3162*, 2014.

[185] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[186] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2130–2137. IEEE, 2009.

[187] Yair Weiss, Rob Fergus, and Antonio Torralba. Multidimensional spectral hashing. In *Computer Vision - ECCV 2012 - 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part V*, 2012.

[188] Mohammad Norouzi and David J. Fleet. Minimal loss hashing for compact binary codes. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, 2011.

[189] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3424–3431. IEEE, 2010.

[190] Mohammad Rastegari, Ali Farhadi, and David A. Forsyth. Attribute discovery via predictable discriminative binary codes. In *ECCV (6)*, 2012.

[191] Yair Weiss, Rob Fergus, and Antonio Torralba. Multidimensional spectral hashing. In *Computer Vision–ECCV 2012*, pages 340–353. Springer, 2012.

[192] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[193] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[194] Hervé Jégou, Matthijs Douze, Cordelia Schmid, et al. Searching with quantization: approximate nearest neighbor search using short codes and distance estimators. 2009.

[195] Genevieve Patterson and James Hays. Sun attribute database: Discovering, annotating, and recognizing scene attributes. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2751–2758. IEEE, 2012.

[196] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[197] Mohammad Rastegari, Jonghyun Choi, Shobeir Fakhraei, Daume Hal, and Larry Davis. Predictable dual-view hashing. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1328–1336, 2013.

[198] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[199] Mohammad Rastegari, Shobeir Fakhraei, Jonghyun Choi, David W. Jacobs, and Larry S. Davis. Comparing apples to apples in the evaluation of binary coding methods. *CoRR*, 2014.

[200] Mohammad Norouzi and David J Fleet. Cartesian k-means. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3017–3024. IEEE, 2013.

[201] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. 2014.

[202] Lorenzo Torresani, Martin Szummer, and Andrew Fitzgibbon. Efficient object category recognition using classemes. In *Computer Vision–ECCV 2010*, pages 776–789. Springer, 2010.

[203] Dan Greene, Michal Parnas, and Frances Yao. Multi-index hashing for information retrieval. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 722–731. IEEE, 1994.

[204] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

[205] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.