ABSTRACT

Title of Dissertation: EFFICIENT IMAGE AND VIDEO REPRESENTATIONS FOR RETRIEVAL

Sravanthi Bondugula, Doctor of Philosophy, 2016

Dissertation directed by: Professor Larry S. Davis
Department of Computer Science

Image (Video) retrieval is an interesting problem of retrieving images (videos) similar to the query. Images (Videos) are represented in an input (feature) space and similar images (videos) are obtained by finding nearest neighbors in the input representation space. Numerous input representations both in real valued and binary space have been proposed for conducting faster retrieval. In this thesis, we present techniques that obtain improved input representations for retrieval in both supervised and unsupervised settings for images and videos.

Supervised retrieval is a well known problem of retrieving same class images of the query. We address the practical aspects of achieving faster retrieval with binary codes as input representations for the supervised setting in the first part, where binary codes are used as addresses into hash tables. In practice, using binary codes as addresses does not guarantee fast retrieval, as similar images are not mapped to the same binary code (address). We address this problem by presenting an efficient supervised hashing (binary encoding) method that aims to explicitly map all the images of the same class ideally to a unique binary code. We refer to the binary codes of the images as 'Semantic Binary Codes' and the unique code for all same class images as 'Class Binary Code'. We also propose a new class based Hamming metric that dramatically reduces the retrieval times for larger databases, where only hamming distance is computed to the class binary codes. We also propose a Deep semantic binary code model, by replacing the output layer of a popular convolutional

Neural Network (AlexNet) with the class binary codes and show that the hashing functions learned in this way outperforms the state of the art, and at the same time provide fast retrieval times.

In the second part, we also address the problem of supervised retrieval by taking into account the relationship between classes. For a given query image, we want to retrieve images that preserve the relative order i.e. we want to retrieve all same class images first and then, the related classes images before different class images. We learn such relationship aware binary codes by minimizing the similarity between inner product of the binary codes and the similarity between the classes. We calculate the similarity between classes using output embedding vectors, which are vector representations of classes. Our method deviates from the other supervised binary encoding schemes as it is the first to use output embeddings for learning hashing functions. We also introduce new performance metrics that take into account the related class retrieval results and show significant gains over the state of the art.

High Dimensional descriptors like Fisher Vectors or Vector of Locally Aggregated Descriptors have shown to improve the performance of many computer vision applications including retrieval. In the third part, we will discuss an unsupervised technique for compressing high dimensional vectors into high dimensional binary codes, to reduce storage complexity. In this approach, we deviate from adopting traditional hyperplane hashing functions and instead learn hyperspherical hashing functions. The proposed method overcomes the computational challenges of directly applying the spherical hashing algorithm that is intractable for compressing high dimensional vectors. A practical hierarchical model that utilizes divide and conquer techniques using the Random Select and Adjust(RSA) procedure to compress such high dimensional vectors is presented. We show that our proposed high dimensional binary codes outperform the binary codes obtained using traditional hyperplane methods for higher compression ratios.

In the last part of the thesis, we propose a retrieval based solution to the Zero shot event classification problem - a setting where no training videos are available for the event. To do this, we learn a generic set of concept detectors and represent

both videos and query events in the concept space. We then compute similarity between the query event and the video in the concept space and videos similar to the query event are classified as the videos belonging to the event. We show that we significantly boost the performance using concept features from other modalities.

EFFICIENT IMAGE AND VIDEO REPRESENTATIONS FOR RETRIEVAL

by

Sravanthi Bondugula

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

Advisory Committee:
Professor Larry S. Davis, Chair/Advisor
Professor Ramani Duraiswami
Professor David Mount
Professor V.S. Subrahmanian
Professor Louiqa Raschid, Dean's Representative

Acknowledgments

I owe my gratitude to all the people who have made this thesis possible and because of whom my graduate experience has been a learning and life changing experience.

First and foremost I'd like to thank my advisor, Professor Larry S. Davis for training me to become a better person and also giving me independence to choose the problems that I am comfortable with. He has always been patient to edit my drafts, helped me with improving my presentation skills and more importantly, solved the problems with me and pointed me in the right direction when I am stuck.

I owe my deepest thanks to my family - my mother, father, sister and husband who have always stood by me and without whom, I would not have done this. I am indebtful to them for all the support, strength and courage they have provided me to complete this journey.

I would also like to thank my roommates/friends - Manisha Ganeshan, Shivangini Pachauri and Soumya Rastogi, who have made the graduate life experience and the career path comfortable and welcoming. I cannot give enough credit to my close friends - Pooja Dasari, Aswani Dhulipalla and Priyanka Tangudu for bearing with me, listening to the graduate success and failure stories and also have patiently waited and forgiven me for all the meetings that I missed.

I also would like to thank all my co-authors - Varun Manjunatha, David Doermann, Pradeep Natarajan, Shuang Wu, Florian Luisier and working with them was truly a learning experience.

I would like to acknowledge financial support from the NSF and MURI Projects, for all the projects discussed herein.

Lastly, thank you all and thank you GOD.

Contents

List of Figures

Chapter 1: **Introduction**

Image (Video) retrieval is an interesting problem of retrieving images (videos) similar to the query. Images (Videos) are represented in an input (feature) space and similar images (videos) are obtained by finding nearest neighbors in the input representation space. This has been a challenging task over decades due to the proliferation of available visual data on the web. To alleviate this problem, efficient input representations have been proposed for visual data in both real valued and binary space to facilitate faster retrieval and provide efficient storage. But, input representations learned for one retrieval setting may not be suitable for others. Therefore, these representations need to be tailored for different objectives of retrieval. For example, binary codes that outperform for an unsupervised setting need not show similar trend in a supervised setting. In an unsupervised setting, binary codes preserve similarity in the Euclidean space of the original features, while in a supervised setting, binary codes preserve similarity in the semantic space (given by the class labels) of images. The performance gap is due to the fact that Euclidean neighbors simply need not be semantic neighbors. Similarly, encoding schemes suitable for compressing small dimensional features cannot be readily extended to high dimensional features due to computational challenges. In this thesis, we address

the problem of learning efficient input representations for some of the less explored retrieval settings and its practical aspects and show that we remarkably improve the performance over the existing state-of-the art retrieval methods.

Specifically, we want to improve retrieval of images and videos by learning efficient binary and concept representations respectively. In particular, we focus on learning improved binary codes in the first three parts of the thesis, while we learn concept based representations for videos in the last part. In the first part, we address the practical aspects of achieving faster retrieval with binary codes for a supervised setting, where the task is to retrieve images of the same class of the given query image. We also propose a deep based model and show superior results over the state of the art. We show that we achieve faster retrieval times with the proposed class hamming distance metric. In the second part, we also propose a supervised learning method that takes into account the relationship between classes using output embeddings-which are vector representation of classes. Our hashing scheme learns such relationship aware binary codes by minimizing the difference between inner product of the binary codes and similarity of the classes. We also introduce new evaluation metrics that consider retrieved images of related classes and show significant gains over the state of the art. In the third part, we learn binary codes for large dimensional vectors like Fisher Vector and Vector of Locally Aggregated Descriptors (VLAD) using hyperspherical hashing functions than adopting the traditional hyperplane hashing functions. We show that the codes learned using hyperspherical hashing functions obtain compact codes and yield better performance than the traditional hyperplane based hashing methods. The last part is different from the

previous works, but embodies the retrieval principles for classification. This work specifically addresses the event classification of videos in zero-shot settings, where no training examples associated with the event are known prior to the classification task. We use a concept based video representation to solve this. Further, we show that multimodal fusion of the scores using visual, audio and text concepts boost remarkably the performance of our system. The outline is given below.

1.1 Outline

In Chapter 2, we address the practical aspects of achieving faster retrieval with supervised hashing by mapping all images of the same class to a unique binary code. In Chapter 3, we learn relationship aware binary codes that rank related class images before unrelated class images when same class images of the query are not retrieved. In Chapter 4, we present a hierarchical approach to compress large dimensional vectors using hyperspherical hashing functions. In Chapter 5, we present a complete zero-shot classification system for videos, solved in a retrieval setting.

- **Semantic Binary Codes:** Fast image retrieval is required for many applications like Image Search and Shopping, especially for large datasets. Hashing addresses this problem by learning compact binary codes for images and using them as direct addresses into hash tables. In practice, using binary codes as addresses does not guarantee fast retrieval, as similar images are not mapped to the same binary code (address). We address this problem by presenting an

efficient supervised hashing method that aims to explicitly map all the images of the same class ideally to a unique binary code. We refer to the binary codes of the images as 'Semantic Binary Codes' and the unique code for all same class images as 'Class Binary Code'. We formulate this intuitive objective 'directly' by minimizing the squared error criterion between the semantic binary codes and the corresponding class binary codes. We further propose a Deep Semantic Binary Code model that utilizes the class binary codes and show that we significantly outperform the state-of-the-art. We also propose a class-based Hamming metric that dramatically reduces the retrieval times for larger databases and also improves the performance of the method by large margins on Cifar-100, Caltech-256 and ILSVRC2010 datasets.

- **SHOE: Supervised Hashing with Output Embeddings:** In this work, we present a supervised binary encoding scheme for image retrieval that learns projections by taking into account similarity between classes obtained from output embeddings. Our motivation is that binary hash codes learned in this way improve both the visual quality of retrieval results and existing supervised hashing schemes. We employ a sequential greedy optimization that learns relationship aware projections by minimizing the difference between inner products of binary codes and output embedding vectors. We develop a joint optimization framework to learn projections which improve the accuracy of supervised hashing over the current state of the art with respect to standard and sibling evaluation metrics. We further boost performance by

applying the supervised dimensionality reduction technique on kernelized input CNN features. Experiments are performed on three datasets: CUB-2011, SUN-Attribute and ImageNet ILSVRC 2010. As a by-product of our method, we show that using a simple k-nn pooling classifier with our discriminative codes improves over the complex classification models on fine grained datasets like CUB and offer an impressive compression ratio of 1024 on CNN features.

- **Hierarchical Spherical Hashing for Compressing High Dimensional Vectors:** High dimensional vectors like Fisher Vectors and Vector of Locally Aggregated Descriptors (VLAD) have been shown to be effective for many computer vision applications like classification, detection including retrieval. However, due to their storage complexity - it becomes intractable to use these features for retrieval. In this work, we address the problem of compressing such high dimensional vectors to high dimensional binary codes in an unsupervised setting. We present a hierarchical approach to compress large dimensional vectors using hyperspherical hashing functions. We provide a practical solution for learning hyperspherical hashing functions by partitioning the vectors and learning hyperspheres in subspaces. Our method is an efficient way to preserve the hashing properties of sub-space hashing functions to generate the full-hashing functions in a divide and conquer fashion. We demonstrate the performance of our approach on the ILSVRC2010 Validation dataset and two large scale datasets: ILSVRC2010 Train and Holidays+Flickr 1M with high dimensional representations of size 128000, 25600 and 12800 respectively.

Our results highlight the compact nature of hyperspherical hashing functions which significantly outperform the state-of-the art methods at compression ratios of 512, 256 and 128. Furthermore, we boost the retrieval performance by introducing an assymetric distance for spherical hashing functions.

- **Zero-shot Event Detection using Multi-modal Fusion of Weakly Supervised Concepts:** Current state-of-the-art systems for visual content analysis require large training sets for each class of interest, and performance degrades rapidly with fewer examples. In this work, we present a general framework for the zero-shot learning problem of performing high-level event detection with no training exemplars, using only textual descriptions. This task goes beyond the traditional zero-shot framework of adapting a given set of classes with training data to unseen classes. We leverage video and image collections with free-form text descriptions from widely available web sources to learn a large bank of concepts, in addition to using several off-the-shelf concept detectors, speech, and video text for representing videos. We utilize natural language processing technologies to generate event description features. The extracted features are then projected to a common high-dimensional space using text expansion, and similarity is computed in this space. We present extensive experimental results on the large TRECVID MED [3] corpus to demonstrate our approach. Our results show that the proposed concept detection methods significantly outperform current attribute classifiers such as Classemes [4], ObjectBank [5], and SUN attributes [43]. Further, we find

that fusion, both within as well as between modalities, is crucial for optimal performance.

Chapter 2: **Semantic Binary Codes**

2.1 Introduction

Approximate Nearest Neighbor(ANN) search for image retrieval using compact binary codes has been extensively investigated for faster retrieval and efficient storage. Encoding images with short bit vectors offers great compression. Retrieval is faster predominantly because binary codes can be used as direct addresses into hash tables and ANN's in principal are identified with just a few lookups, compared to an exhaustive linear scan [6]. In practice, using binary codes as addresses does not guarantee fast ANN search, as similar images are not mapped to the same binary code(address). Instead, nearest neighbors must be discovered from examining binary codes within some Hamming radius around the query code. So, the number of lookups required grows exponentially with radius. State-of-the art hashing algorithms then resort to linear scan for large radius. Therefore, there is an explicit need for a hashing algorithm that learns to map similar images to the same binary code.

There exist supervised [7–10] and unsupervised [11–14] hashing algorithms for learning similarity preserving binary codes. Unsupervised hashing algorithms try to preserve Euclidean neighbors while supervised hashing algorithms try to preserve

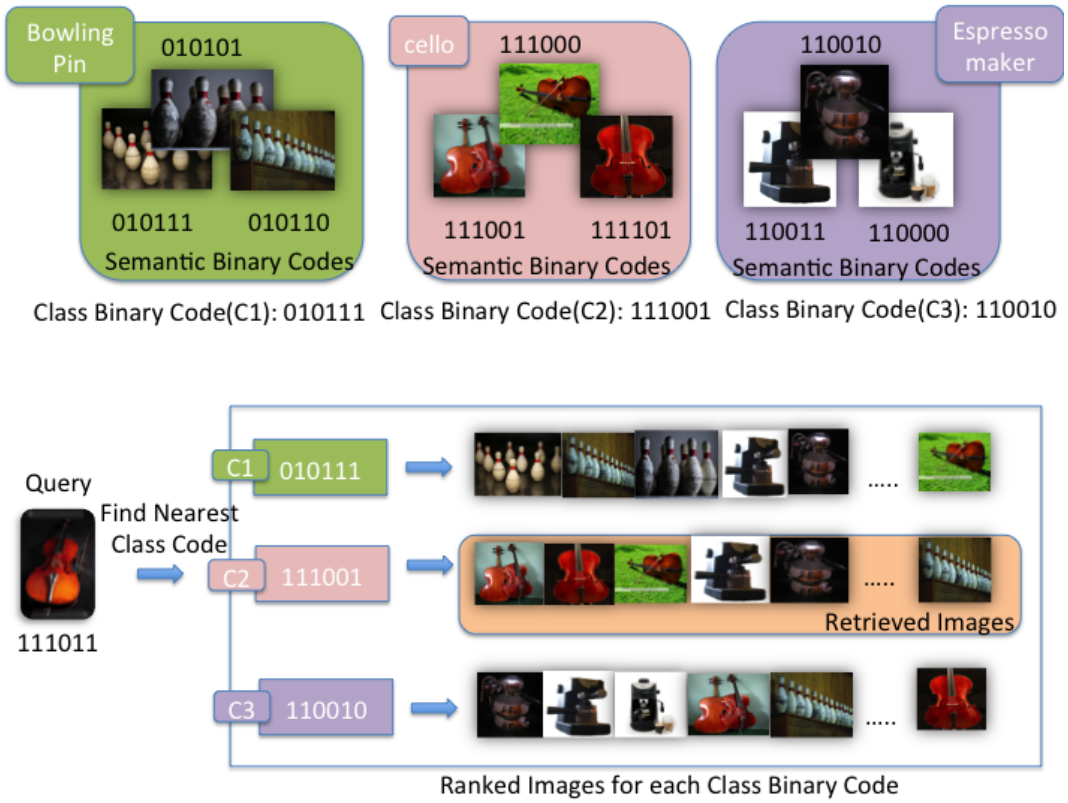

Figure 2.1: Illustration of the Semantic Binary Codes idea and retrieval using Class Binary Codes. Here, instead of searching 9 image binary codes, we search only 3 class binary codes.

semantic neighbors. The notion of similarity is well defined for semantic neighbors compared to Euclidean neighbors as similarity values are discrete(1(same) or 0(different)) for the former and continuous for the latter. In this work, we develop a new approach to supervised hashing, where we want to **learn both image and class binary codes such that all images from the same class are ideally assigned a unique binary code**. We refer to the image binary codes as 'Semantic Binary Codes' (SBC) and the unique binary code for a class as the 'Class Binary

Code' (CBC). Figure 2.1 illustrates the idea of our method. By learning both these codes, the performance of a hashing algorithm is naturally improved by retrieving more relevant neighbors within a smaller Hamming ball around the query code.

Only a handful of approaches [7–9, 15, 16] have been developed to learn a unique binary code for all images from the same class, but they do not formulate this directly into the objective. They instead, learn same binary codes for same class images and different binary codes for different class images. Further, their optimization is complex due to the pair-wise formulation which forces them to resort to a greedy iterative procedure of learning bit by bit which leads to a slow training process. In contrast to these approaches, we propose a simple and efficient supervised hashing algorithm with point-wise formulation of the objective. The first objective eliminates noise by learning projections of input features such that the projected vectors are highly correlated with class vectors. The second objective then minimizes the quantization error between semantic and class binary codes. We jointly optimize these two objectives to obtain the hash functions that meet our goal. Further, we propose a Deep Semantic Binary Code model that utilizes the CBC's.

The contributions of our work are as follows: 1. To the best of our knowledge, our approach is the first 'direct' attempt to learn Semantic Binary Codes(a unique binary code for all semantically similar images) and Class Binary Codes by formulating an intuitive objective function. 2. We present a novel joint learning framework for simultaneous dimensionality reduction and semantic binary codes. Using a least squares formulation for both objectives, we obtain a simple objective function which has an efficient closed-form solution. 3. We also propose a new

metric: Class Hamming Distance(CHD). Using CHD, we retrieve results for a query by only calculating Hamming Distance to the class binary codes, rather than calculating Hamming Distance to all the database codes, leading to a large decrease in retrieval times. 4.We also demonstrate superior performance and achieve faster training and retrieval times compared to the state of the art shallow supervised hashing techniques for Caltech-256, Cifar-100 and ILSVRC2010 Train datasets. The Deep Semantic Binary Code model proposed also outperforms the state of the art Deep Supervised Hashing techniques.

The remainder of the work is organized as follows. We first discuss related work in Section 2.2. We propose a novel joint framework for dimensionality reduction and learning semantic and class binary codes and its deep learning model in Section 2.3. Retrieval experiments are carried out in Section 4.4. Finally, we conclude in Section 2.5.

2.2 Related Work

Supervised hashing methods in [7–9, 15, 17–19] use pair-wise formulations that lead to complex optimization procedure. Liu et al ([7]) proposed the Supervised Hashing with kernels(KSH) method that minimizes the difference between inner products of the binary code and similarity for pairs of images. Similarity is 1 (-1) for same (dissimilar) class pairs. They obtain a simple, clean objective function unlike the complicated objective functions in MLH [15] and BRE [20]. FastHash [8] applies a two-step procedure [17] to solve the KSH loss function: First, they learn the binary

codes using Graph cuts and then learn the parameters of the hash function with Boosted Decision Trees. They accommodate training of large numbers of samples with high dimensional features and show better performance than KSH [7]. Recently, Ge et al [9] introduced the Graph Cuts Coding(GCC) method which also adopts a two-step approach [17], but iteratively. At each iteration, they first learn the binary codes using GraphCuts and then learn the hash functions using linear SVMs. Rastegari et al [18] propose learning predictable and discriminative hash functions using a linear or kernel SVM formulation using bits as labels. For each hash function, they minimize(maximize) the distance between the codes of same(dissimilar) class images. All these hashing methods obtain similar codes for same class images by minimizing(maximizing) intra(inter) class distance between image codes but not between class and image codes.

Recently, Wang et al [19] construct compact binary codes for both images and tags such that the observed tags are consistent with the constructed binary codes. They minimize three critiera: difference between similar tag and image codes, difference between similar image codes and difference between similar tag codes. This is highly suitable for datasets with weakly labeled information which require fast image tagging. Note that the binary codes for tags are learned but not the class codes. Very recently, Supervised Discrete Hashing(SDH) [10] learns binary codes that are optimal for classification. They employ a joint optimization framework which jointly learns a binary embedding and a linear classifier. SDH also obtains a simple objective function that yields a closed form solution by incorporating a Discrete-Coordinate Descent method to learn one bit at a time. While, our end goal

is not specifically to learn better classification codes, we intend to learn semantic binary codes and binary representations for classes.

We also review several deep supervised hashing methods [16, 21–25] that have been proposed and showed superior performance over the shallow methods. Semantic Hashing(SH) [6], one of the earliest works, uses Restricted Boltzmann Machines(RBMs) to train Binary Auto Encoders. However, SH did not do well due to limited computational power available then. The following Deep Supervised Hashing(DSH) Models: Sparse Similarity-Preserving Hashing (SSPH) [21], Deep Hashing (DH) [22], Convolutional Neural Network Hashing (CNNH) [16], Deep Semantic Ranking Hashing (DSRH) [23], Deep Neural Network Hashing (DNNH) [24] and Deep Learning of Binary Hash Codes for Fast Image Retrieval (DLH) [25] have been proposed. The output of these neural networks yields a binary code by applying either a hyperbolic tangent or a sigmoid function to the output followed by a thresholding operation, leading to the final binary hash code. While, some networks like SSPH, DH assume a hand crafted visual representation to learn binary codes that minimizes the reconstruction error. These methods are limited by the performance of the assumed hand crafted visual representation. To overcome this, CNNH, DSRH, DNNH and DLH networks do not assume an specific feature representation and instead, integrate both the feature learning and hash value learning in the optimization. The later methods have subsequently shown better performance. For a comprehensive related work on DSH schemes, we refer the readers to the short survey by Liu[1]. We also present a deep extension of the SBC model, Deep Semantic

[1] http://www.ee.columbia.edu/w̃liu/WeiLiu_DLHash.pdf

Binary Code model, where we learn a deep neural network that utilizes the class binary codes.

2.3 Semantic Binary Codes

Input features/labels: We are given a data set of n samples $\{(\boldsymbol{x_i}, y_i)\}_{i=1}^{n}$, where $\boldsymbol{x_i} \in \mathcal{R}^d$ is the input feature and $y_i \in \{1, 2, ..k\}$ denotes the class label of the i^{th} sample. Let $Y \in \mathcal{R}^{k \times n}$ denote the class indicator matrix with one-of-k embeddings i.e $Y_{ij} = 1$ if the class of the j^{th} sample is i and -1 otherwise. We apply the radial basis kernel features with m random sample points to obtain $\phi(x) : x \rightarrow \bar{x} \in \mathcal{R}^m$ as done commonly in [7, 10, 26].

Objective: We aim to jointly minimize both the dimensionality reduction error and quantization error between the image binary codes and the corresponding class binary codes.

Input Projections/Hash Projections: We want to learn semantic binary codes $\boldsymbol{b_i} \in \{-1, 1\}^l$ and the corresponding class binary codes $\boldsymbol{c_{y_i}} \in \{-1, 1\}^l$, both of length l for the i^{th} sample, such that a unique binary code is learned for all same class images. We utilize the hash functions $\{h_j(\phi(\boldsymbol{x})) = sgn(\boldsymbol{u_j}^T \phi(\boldsymbol{x})\}_{j=1}^{l}$ to obtain l bits, where, sgn(.) is the sign function, which outputs +1 for positive number and -1 otherwise. $\boldsymbol{u_j} \in \mathcal{R}^m$ is a column vector that gives the bit b_{ij} for the i^{th} sample. Let $U = [\boldsymbol{u_1}, \boldsymbol{u_2}, ..., \boldsymbol{u_l}] \in \mathcal{R}^{r \times l}$ which can be decomposed into two projection matrices V and W. Then, the corresponding hash functions are: $\{h_j(\phi(\boldsymbol{x})) = sgn(\boldsymbol{w_j}^T V^T \phi(\boldsymbol{x})\}_{j=1}^{l}$, where $V \in \mathcal{R}^{m \times r}$ are the input projections and

$\boldsymbol{w_j} \in \mathcal{R}^r$ are the hash projections on the reduced features $V^T\phi(\boldsymbol{x})$. We solve the following joint formulation of the objective:

$$\min_{W,V,c} \sum_{i=1}^{n} \left\| V^T\phi(\boldsymbol{x_i}) - Y_i \right\|_2^2 + \sum_{i=1}^{n} \left\| \boldsymbol{b_i} - \boldsymbol{c_{y_i}} \right\|_2^2$$

$$b_i = sgn(W^T V^T \phi(x_i)) \tag{2.1}$$

where, $\|.\|_2^2$ is the L2-norm.

The first term of the objective eliminates noise by learning projections of input features such that the projected vectors are highly correlated with the class vectors. This is a least squares error formulation for dimensionality reduction and has been shown to be equivalent to CCA [27], with an appropriate form of class indicator matrix $((YY^T)^{-\frac{1}{2}}Y)$. Similarly, the second term of the objective decreases the quantization error between image and class binary codes. Relaxing the sign function in the objective, we get:

$$\min_{W,V,C} \left\| V^T\phi(X) - Y \right\|_F^2 + \left\| W^T V^T \phi(X) - C \right\|_F^2$$

$$B = sgn(W^T V^T \phi(X)) \tag{2.2}$$

where, $\|.\|_F^2$ is the Frobenius norm and $C \in \{-1, 1\}^{l \times n}$ with $C_i = c_{y_i}$ denote the corresponding class codes for all the samples. Solving this is an NP-Hard problem as there are many variables to learn. We therefore employ an iterative procedure, by solving for one variable at a time.

Solving for V***:*** Given the binary codes of classes C and the hash projections W, we can solve for the input projections V. Let $S = V^T\phi(X)$, we first learn S and then learn V. Expanding (2.2), we obtain:

$$tr(S^T(I + WW^T)S) - 2tr(S^T(WC + Y)) + nl + const$$

where tr(.) is the Trace function. Taking derivative of the above equation w.r.t.to S and equating to zero yields:

$$S = (WW^T + I)^{-1}(WC + Y) \tag{2.3}$$

Given the reduced features S, we then obtain the input projections V as:

$$V = (\phi(X)\phi(X)^T)^{-1}\phi(X)S^T \tag{2.4}$$

Now, we can learn binary codes of images B and classes C independently from the dimensionality reduction step, by solving:

$$\min_{W,C} \left\|W^T S - C\right\|_F^2$$

$$s.t.\ B = sgn(W^T S) \tag{2.5}$$

Solving for W***:*** Given binary codes of classes C, we obtain the hash projections W on the reduced features S by taking the derivative of (2.5) and setting it to zero. We obtain the following closed form solution for W:

$$W = (SS^T)^{-1}SC^T \tag{2.6}$$

Utilizing the hash projections learned, we now obtain the binary codes for the images B by simply taking the sign of the projected vector $W^T S$, given as $B = sgn(W^T S)$.

Solving for C***:*** Given binary codes of the images B, we can learn the class binary codes without the relaxed formulation. We rewrite the objective in (2.5) with discrete constraints on binary codes, for each sample as:

$$\min_{\boldsymbol{c}} \sum_{i=1}^{n} \|\boldsymbol{b_i} - \boldsymbol{c_{y_i}}\|_2^2, \ \ s.t.\ \boldsymbol{c_{y_i}} \in \{-1, 1\}^l \tag{2.7}$$

Set/Tr(t)	SBC(C)	SBC	SDH	ITQ	FastH	KSH-
Cal256	31	31	408	**10**	2413	2e+5
Cifar100	44	44	378	**8**	739-	5800
Ilsvrc10	116	116	283	**81**	8580-	3e+4
Set/R(t)	SBC(C)	SBC	SDH	ITQ	FastH	KSH
Cal256	**0.25**	1.4	1.4	1.4	2.6	1.4
Cifar100	**0.13**	2.9	2.9	2.7	4.8	2.9
Ilsvrc10	**0.94**	351	351	350	354	351

Table 2.1: We compare the training(Tr(t) in sec) and retrieval(R(t) in milli sec) times on Caltech256(128 bits), Cifar-100(96 bits) and ILSVRC2010(256 bits). (C) denotes that, we use CHD.

It is easy to see that the objective is minimized when the class code is obtained by simply taking the sign of the sum of the bit vector of the samples. To see this, let $\mathcal{Y}_\kappa$ contain the indices of the samples belonging to class κ and $n_\kappa = |\mathcal{Y}_\kappa|$, the number of samples per class. We can now rewrite the objective function independently for each class:

$$\min_{\boldsymbol{c}} \sum_{j \in \mathcal{Y}_1} \|\boldsymbol{b_j} - \boldsymbol{c_1}\|_2^2 + \sum_{j \in \mathcal{Y}_2} \|\boldsymbol{b_j} - \boldsymbol{c_2}\|_2^2 + ... + \sum_{j \in \mathcal{Y}_k} \|\boldsymbol{b_j} - \boldsymbol{c_k}\|_2^2 \tag{2.8}$$

For a single class, Expanding the objective, we get:

$$\begin{aligned} \min_{\boldsymbol{c_\kappa}} \sum_{j \in \mathcal{Y}_\kappa} \|\boldsymbol{b_j} - \boldsymbol{c_\kappa}\| &= (\sum_{j \in \mathcal{Y}_\kappa} \|\boldsymbol{b_j}\|) + n_\kappa \|\boldsymbol{c_\kappa}\| - 2(\sum_{j \in \mathcal{Y}_\kappa} \boldsymbol{b_j^T})\boldsymbol{c_\kappa} \\ &= n_\kappa l + n_\kappa l - 2\tilde{\boldsymbol{B}}_{\boldsymbol{\kappa}}^{\ T} \boldsymbol{c_\kappa} \end{aligned} \tag{2.9}$$

Method	#Train	#anchor	mAP	preH@0	pre@1
SBC(C)	2560	512	40.2	17.5	54.49
SBC(C)	6400	1280	54.7	47.8	64.02
SBC(C)	6400	2560	62.7	60.11	66.03
SBC(C)	12800	2560	**63.64**	**59.73**	**66.56**
SBC	12800	2560	59.63	10.59	65.33
SDH	12800	2560	55.4	18.3	65.13
ITQ+CCA	12800	-	49.44	4.7	59.7
FastHash	12800	-	55.85	0.1	59.8
KSH	6400	1280	46.29	11.6	59.8
BRE$^-$	6400	-	15.98	0.1	40.4

Table 2.2: Comparing various metrics and methods for 128 bits on Caltech-256, except for BRE, where we report performance with 64 bits as the method did not converge in 8 hours for 128 bits.

where $\tilde{\boldsymbol{B}}_{\boldsymbol{\kappa}} = \sum_{j \in \mathcal{Y}_\kappa} \boldsymbol{b_j}$. As $\boldsymbol{c_\kappa}$ is a binary code $\in \{-1, 1\}^l$, it is easy to infer that (2.9) is minimized when the product $\tilde{\boldsymbol{B}}_{\boldsymbol{\kappa}}^{\boldsymbol{T}} \boldsymbol{c_\kappa}$ is maximized. This happens only when the class code $\boldsymbol{c_\kappa}$ takes the same sign as the sum of the bit vector $\tilde{B}_\kappa$. In other words, each bit of the class code is given by the maximum voted bit of all the samples of the class or equivalently, the centroid of the binary codes of the class samples. Given binary code vectors, we have:

$$\boldsymbol{c_\kappa} = sgn(\sum_{j \in \mathcal{Y}_\kappa} \boldsymbol{b_j}), \quad \forall \kappa \tag{2.10}$$

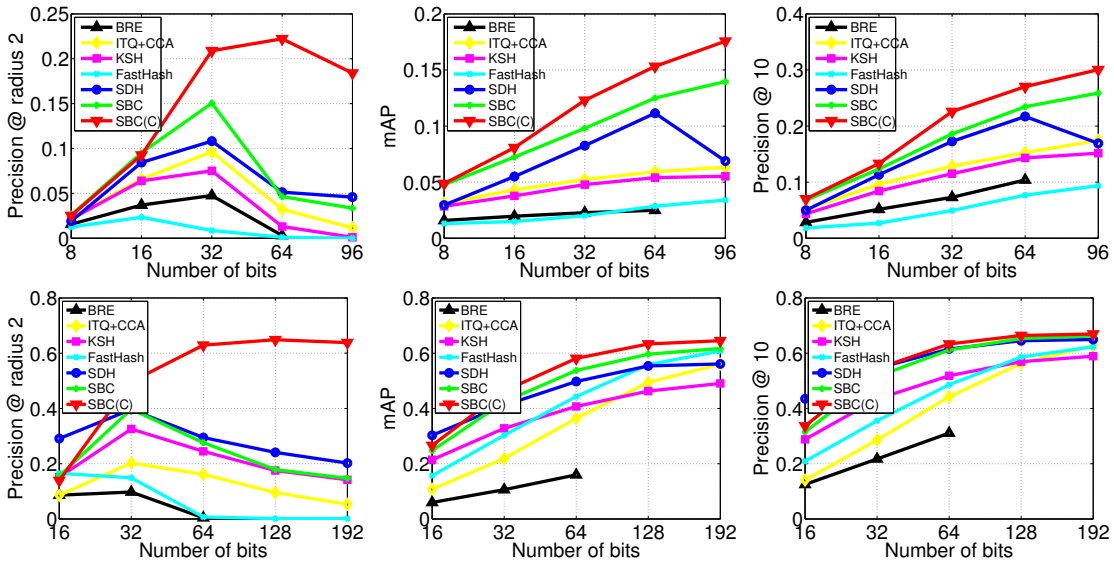

Figure 2.2: The top row and the bottom row shows the performance of various metrics for varying numbers of bits on Cifar-100 and Caltech-256 datasets respectively.

We have now learned all the variables, one at a time. We do this iteratively until the objective converges or we reach a maximum number of iterations. We empirically observed that the whole algorithm converges very fast and typically takes fewer than 3 iterations to converge. In either case, we set the maximum number of iterations to be 5. We use initialization from CCA to learn the initial W and then obtain an initial B & C, using the initial solution of V as $(\phi(X)\phi(X)^T)^{-1}\phi(X)Y^T$. The detailed SBC algorithm is given in Algorithm 1.

2.3.1 Class Indicator Matrix Y

Given a mean centered Y matrix, Let $H = (YY^T)^{-\frac{1}{2}}Y$. [27] shows that when $Y = H$, solving equation (2.11) is equivalent to solving the CCA problem or an Orthonomal Partial Least Squares problem under the following criteria: 1) The

minor condition that rank(X) = n-1 or rank(Y) = k is satisfied, which generally holds true for high dimensional data and 2) The final goal is to do nearest neighbor search or train SVMs on the projected features. Since, these criteria hold in our case, we use the class indicator matrix $Y = H$ and refer to the DR-step as LS-CCA.

2.3.2 Initialization of V

We obtain the initial solution for V by solving the LS-CCA using (2.11).

$$\min_{V} \left\| V^T \phi(X) - Y \right\|_F^2 \tag{2.11}$$

where $\|\|_F^2$ is the Frobenius norm and $Y = H$, given in 2.3.1. Solutions of equation (2.11) are given by ridge regression as:

$$V = (\phi(X)\phi(X)^T)^{-1}\phi(X)Y^T \tag{2.12}$$

2.3.3 Class Hamming Distance(CHD)

Most hashing methods that employ the hyperplane hashing function form: $h(x) = sgn(U^T x)$ use Hamming Distance(HD) between the query code (b_q) and database codes (b_d) to compute nearest neighbors in the database for a query(q). Hamming distance for binary codes $b_q, b_d \in \{0,1\}^l$ is given as, $HD(b_q, b_d) = |b_q \oplus b_d|$, where $\oplus$ is the XOR operation and $|.|$ denotes the number of +1 bits in a given binary code. Since, we learn class binary codes, we propose a suitable metric: class-based Hamming Distance(CHD).

Given binary codes for each class, CHD retrieves nearest neighbors for a query q in two steps: First, we compute the nearest class binary code c i.e. $c = c_\kappa, \kappa =$

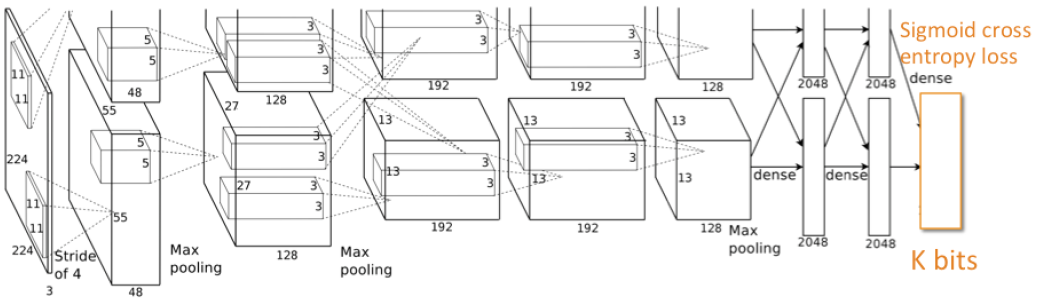

Figure 2.3: Neural Network architecture for the Deep Semantic Binary code model.

$\arg\min_j HD(c_j, b_q)$. Second, we retrieve the nearest neighbors in the database to the class binary code c (instead of the test binary code b_q). Note that, we do not use any class information when we rank the nearest neighbors in the database to the class binary code.The advantages of doing this are two fold: 1) For a correctly assigned class binary code, there is a high probability of always obtaining the same class images as the top retrievals, but the same cannot be said for an incorrect assignment. We expect to reduce the overall error using this approximation. 2) Retrieval is much faster as we only need to compute Hamming distance to the class codes. For each class code, nearest neighbors in the database can be computed offline once and the ranking is stored. Nearest neighbors can now be returned with a single lookup. The total retrieval time is only $O(k)$ instead of $O(n)$, $k << n$ (assuming, linear search).

2.3.4 Deep Semantic Binary Codes

Recently, Deep Supervised Hashing(DSH) methods [16,24,25] have been shown to improve hashing performance by large margins over shallow methods([7,28]). The model of SBC is a single layer neural network and does not take advantage of the deep training procedure. We present a deep neural network method for SBC that also aims to learn a unique binary code for all same class images. We do so by training a deep network model using the Class Binary Codes learned to form the output layer as shown in Figure 2.3. We retrain the AlexNet [29] model by replacing the 1000-dimensional output layer(i.e. fc8) with the Class Binary Codes(Equation (2.10)) of the images and use Sigmoid Cross Entropy Loss as the loss function for the output layer. For a new image, hash codes are obtained by thresholding the sigmoid activations of the output ($thr = 0.5$). We refer to this model as a Deep Semantic Binary Code model and the binary codes learned from the network as Deep Semantic Binary Codes(SBC-D). We expect that the SBC-D model learns to preserve the semantic similarity between the same class images by learning non-linear mappings between the image pixels and the class binary codes.

2.4 Experiments

We evaluate our proposed supervised hashing method(SBC) and deep model on three datasets containing large numbers of categories: Caltech-256 [30], Cifar-100 [31] and ILSVRC2010 Train [32]. We compare our approach to the state-of-the art supervised hashing techniques: SDH [10], FastHash [8], KSH [7], ITQ with

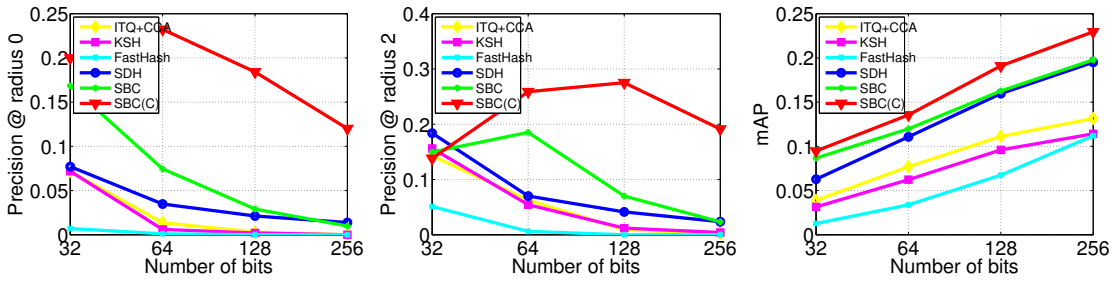

Figure 2.4: Plots showing Precision @ radius 0 , Precision@ radius 2, mAP@10K for the ILSVRC2010 Train dataset.

CCA(ITQ+CCA) [28] and BRE [20]. We use their publicly available implementations and the suggested parameters to obtain their best performance. Caltech-256 dataset consists of 30607 images from 256 categories and 1 background category. We compute 4096 dimensional Convolutional Neural Network (CNN) features from the fc7 layer of AlexNet [29] for each image using the Caffe library developed by [33]. Cifar-100 dataset contains a subset of 60K images from 100 categories of the Tiny 80M dataset [34]. These images are only 32x32 images and so we only compute 512 dimensional GIST features. For the ILSVRC2010 Train dataset containing 1.2 Million Images of 1000 categories, we also compute fc7 CNN features. We kernelize the features which take the form $\{\kappa(x, a_1), \kappa(x, a_2), ...\kappa(x, a_m)\}$ where κ is a radial basis kernel, and m is the cardinality of a subset of training sample, designated as 'anchor points' as done in [7, 10, 26]. All the features are unit normalized.

2.4.1 Train/Test Partitions

Caltech-256 dataset contains around 30607 images from 256 categories and 1 background category. In this dataset, the number of images in each category varies a lot, with a minimum of 80 images in a category and a maximum of around 800 images. To equally weigh the performance of each class, we follow the general testing protocol for Caltech-256 i.e., we choose an equal number of test images(N=25) per category as queries and 50 images per category to form the retrieval set. We choose three subsets of the retrieval set for training, correspondingly $\{10, 25, 50\}$ images per category to form train sets of 2560, 6400 and 12800 images. We compute the Radial Basis Function(RBF) features $\phi(x)$ with varying numbers of anchor points: $m = \{512, 1280, 2560\}$. Cifar-100 constitutes images of 100 classes, with each class containing 600 samples. The entire dataset is split into a test set with 1000 samples(10 per class) and a retrieval set with all remaining samples. We report performance for methods trained with subsets of size 5000 and the whole retrieval set and construct kernelized features with anchor points of size 1000 and 3000. For the ILSVRC2010 Train dataset containing 1.2M images of 1000 categories, we randomly select 1000 queries and use the rest as the retrieval dataset. For training, we chose 50K samples(50 per class) from the retrieval set and again compute RBF features with 3000 anchor points. For all the datasets, we construct the ground truth from class label information, required for supervised training. During retrieval, we do not use any class label information.

2.4.2 Evaluation metrics

We report the standard retrieval performance metrics: Precision@K(pre@K), Mean Average Precision(mAP) and Precision@Hamming radius (preH@r). pre@K gives the precision at top-K nearest neighbors retrieved. Here, we report for $K = 10$ for both Caltech-256 and Cifar-100 datasets. mAP calculates the area under the precision-recall curve, thereby evaluating the performance of the method for all the ranked images. For Caltech-256 and Cifar-100, we report mAP and for ILSVRC2010 Train, we report mAP@10K. preH@r is the percentage of the number of images within a Hamming radius r of the query code that belong to the same class as the query code. For $r = 0$, these are exact collisions i.e. images are mapped to the same code as the query code. We penalize the query when there are no retrievals within a Hamming radius r. We report Precision@Hamming radius 2 for Caltech-256 and Cifar-100 datasets. For ILSVRC2010 Train, we report both Precision@Hamming radius 0 and 2.

We show the performance of our method with both Hamming distance and Class-based Hamming distance (CHD). We append the notation (C) when CHD based performance is reported. For Caltech-256, Cifar-100 and ImageNet datasets, we learn $\{16, 32, 64, 128, 192\}$, $\{8, 16, 32, 64, 96\}$ and $\{32, 64, 128, 256\}$ bits respectively.

2.4.3 Comparison with other methods

BRE, KSH and FastHash incur high training times for large training samples. In our experiments, for efficiency reasons, we show performance of these methods for only 6400 for Caltech-256 dataset and 5000 training samples for Cifar-100 dataset . SDH [10] can be solved with both discrete or relaxed constraints on the binary codes in learning the classifier. They show that discrete binary constraints are critical to obtain maximum performance. So, we only show the results of SDH with discrete constraints. Both, SDH and ITQ+CCA are scalable and efficient. They obtain the maximum performance with larger numbers of training samples and therefore, we show their performance when trained on the larger training set.

Table 2.2 compares the performance of our methods learned with various numbers of training samples for 128 bits on the Caltech-256 dataset. We vary the number of anchor points up to 2560 and training samples up to 12800 and show that performance improves with more anchor points and larger training sets. For all the reported metrics, we significantly outperform the state-of-the art methods by a large gap and we do this with very small training and retrieval times as indicated in Table 2.1.

The top row in Figure 2.2 shows performance on the Cifar-100 dataset. Cifar-100 has a smaller number of classes(100) than Caltech-256 but many samples per class. The results show that SBC(C) generalizes better than the second best method SDH on all metrics. Here, we see that as we learn a larger number of bits, SDH performs better than FastHash, KSH, ITQ+CCA and BRE but worse than both

SBC and SBC(C). We also observe that the performance of both SBC (without the Hamming distance) and SDH converges for precision@radius 2 for 96 bits.The bottom row of Figure 2.2 compares the performance on Caltech-256 dataset for various numbers of bits. We observe that SBC(C) shows superior performance consistently over all metrics: preH@2, mAP and pre@10. We report performance of BRE only up to 64 bits as it did not converge in 8 hours for more bits.

We also compare our method on the large dataset (ILSVRC 2010 Train) to demonstrate the scalability and efficiency of our method in Figure 2.4. Both, SBC and SBC(C) perform significantly better than the other methods at varying number of bits on all the performance measures. Particularly, there is a larger gap with the second best method SDH for accurately finding nearest neighbors with exact collisions, even for longer bits. For KSH, we chose $10,000$ training samples for 1000 anchor points to keep the training time under 8 hours. We do not show the results of BRE due to its high training times. For FastHash, we observe that the performance is very low with 5K training samples. So, we use 50K samples at the expense of training time. We still see that it does not perform well on preH@0 and 2, as also shown in [10]. Table 2.1 reports the training and retrieval times of our method and the competing methods for Caltech-256, Cifar-100 and ILSVRC2010 Train datasets. For all the datasets, SBC is efficient and scalable in training and so is ITQ. The retrieval times of SBC with CHD is significantly faster than HD, as we only compute Hamming distance to the class binary codes. Figure 2.5 shows the qualitative results.

Mthd	64 bits			96bits		
	mAP	pre@1	preH@0	mAP	pre@1	preH@0
SBC-D	**57.9**	**57.4**	**52.4**	**49.7**	**52.5**	**45.4**
DLH [25]	14.2	26.6	5.8	26.9	42.4	2.7

Table 2.3: Performance comparison of DSH methods for various metrics and various numbers of bits for Cifar-100 dataset.

2.4.4 Comparison Methods for SBC-D:

It has been shown that DLH [25] outperforms the model of CNNH and its variants proposed by Xia et al [16] and other Deep Supervised Hashing methods that preserve semantic similarities ([21], DH [22]). As training deep neural networks is computationally expensive, we here compare SBC-D with only the best baseline: DLH [25]. We here show the results for Cifar-100 with 100 classes and use the same train/test partitions described above.

2.4.5 Experiments for SBC-D:

For training our model, we use the AlexNet [29] architecture and replace the output layer with the learned Class Binary Codes in Section 2.3. For training the model of DLH, we use their publicly available code[2]. For Cifar-100, we use a batch size of 100 and we train two models for learning 64 and 96 bits. We train up to

[2]https://github.com/kevinlin311tw/caffe-cvprw15

Iter	64 bits		96bits	
	SBC-D	DLH	SBC-D	DLH
10,000	32.5	1.0	23.2	1.3
20,000	37.3	4.2	25.3	8.5
30,000	51.1	9.8	38.9	20.2
40,000	55.4	12	45.1	23.4
50,000	**57.9**	14.2	49.7	26.8

Table 2.4: Performance(mAP) comparison of DSH methods for various numbers of iterations for Cifar-100 dataset.

50000 iterations, such that training is completed in a reasonable amount of time. We report the mean average precision(mAP), precision@1(pre@1) and precision@radius 0(preH@0) for both the methods in Table 2.3 at 50K iterations. Table 2.4 compares the performance of SBC-D and DLH for the learned numbers of bits.

We see that our model SBC-D outperforms DLH for all the metrics for both 64 and 96 bits and has significantly improved over SBC. High Precision of exact collision binary codes(preH@0) of our method indicates that we achieve the goal of mapping all same class images to a unique binary code. We report a gain of 43.7% and 23% in mAP over DLH and a gain of 42% and 30.7% over SBC for 64 and 96 bits respectively. We notice that the mAP of SBC-D is lower for 96 bits than 64 bits. We expect that this is because of learning from an over complete representation, which resulted in overfitting. However, notice that the performance

of DLH improved from 64 to 96 bits, but is still significantly lower than our proposed deep model. Table 2.3 suggests that the initial SBC-D models learned(i.e. at few iterations) are also efficient and do better than the best models of DLH for both 64 and 96 bits. Both the methods improve performance with more iterations. It is surprising that the model of DLH did not do well for the Cifar100 dataset as opposed to the competitive results shown for the Cifar-10 dataset in its work. We suspect that this may be because the number of latent functions(i.e. bits) learned are less than the number of classes. We have shown that by transforming the labels to the class binary code space and replacing the output layer with the binary codes, an efficient supervised deep hashing method can be learned. Although, we emphasize that the improvement in performance with the deep model comes with an expense of training time(10 hours to train each of the 64 and 96 models), compared to only few minutes for the SBC model.

2.5 Conclusion

The key idea in our work is to address the practical aspects of faster retrieval using image binary codes as addresses. Our goal was to learn supervised hashing functions such that all same class images are mapped to a unique binary code. We proposed a new supervised hashing algorithm that jointly minimizes the dimensionality reduction error and quantization error between the image and class binary codes, by formulating independent objectives with a least squared criterion. A deep extension of the SBC model is also presented to take advantage of the deep train-

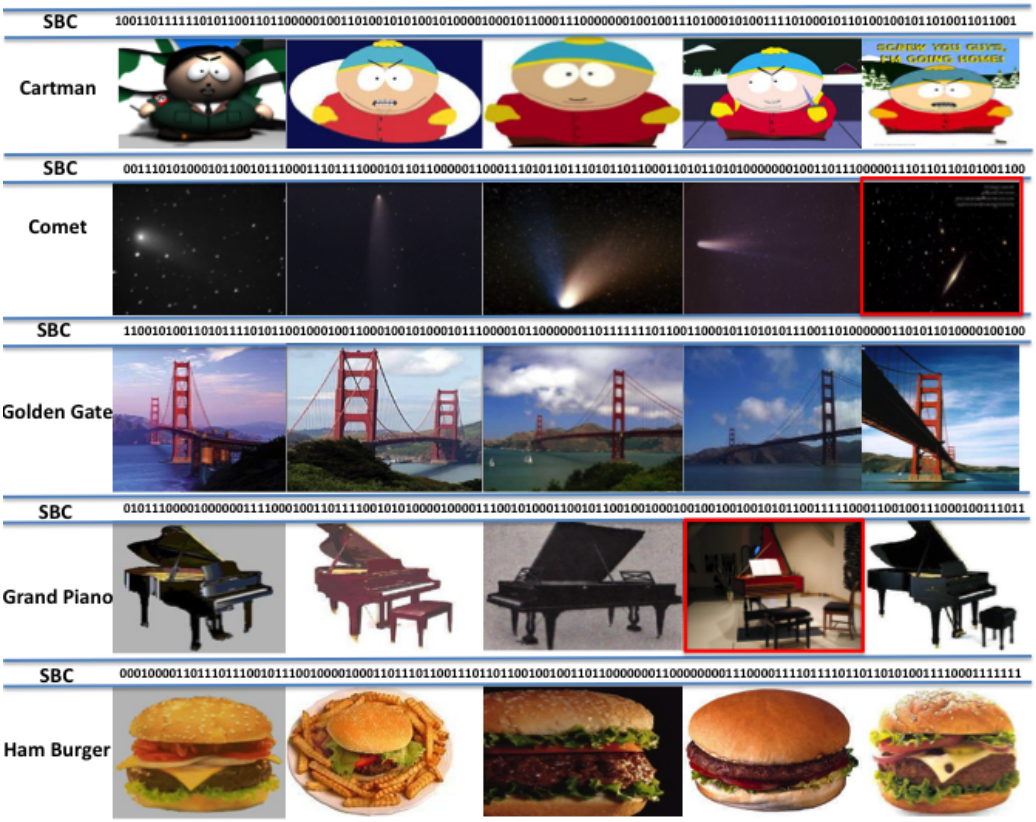

Figure 2.5: Images of Caltech-256 dataset that have the same semantic binary code given by the corresponding 128 bit SBC. For each row, we mention the most common category of the images. False Positives are indicated by red boundaries.

ing procedure. We also introduced Class Hamming Distance that utilizes the class binary codes and shows significant improvement in performance and retrieval times compared to instance-based Hamming Distance. Our experiments demonstrate the superiority of both the shallow and deep proposed methods over the state-of-the art.

Algorithm 1 Learning Semantic Binary Codes(SBC)

Input: Given $\{(x_i, y_i)\}_{i=1}^{n}$ as (input, label) pair, $y_i = \{1, .., k\}$. Let $X \in \mathcal{R}^{d \times n}$ denote the input feature matrix.

Preprocessing: Choose randomly m anchor points and construct $\phi(X) \in \mathcal{R}^{m \times n}$ using RBF kernel. Also, construct a binary one-of-k class indicator matrix: $Y \in \{-1, 1\}^{k \times n}$ with $Y_{ij} = 1$ if the class of the j^{th} sample is i and -1. Mean center the matrix Y.

Equivalent Class indicator matrix for LS-CCA: Construct $H = (YY^T)^{-\frac{1}{2}}Y$.

Goal: To learn l bits i.e. l and r projections for $W \in \mathcal{R}^{r \times l}$ and $V \in \mathcal{R}^{m \times r}$ and class binary codes C.

Initialization:

***V*:** Solve for V using the ridge regression solution in equation (2.12), given $Y = H$ and $\phi(X)$.

***W*:** Given V, apply the DR-mapping: $S = V^T\phi(X)$. Then, learn W using CCA with S and $Y = H$ as input and output modalities.

***C*:** Obtain $B = sgn(W^T S)$ and then C using (2.10)

Optimization:

repeat

- Solve for S and then V using equations (2.3) and (2.4), given $Y = H$ and $\phi(X)$.

- Solve for W using equation (2.6)

- Obtain $B = sgn(W^T S)$ and then c_κ using equation (2.10), $\forall \kappa$. $C_i = c_{y_i}$; for each bit: most occuring bit is assigned

until Convergence Criteria met or max number of iterations

Final Hash Projection: $U = VW$

Query(q) binary code: $binarycode(q) = sgn(U^T\phi(q))$

Class binary codes: Class binary codes given by C.

Chapter 3: **SHOE: Sibling Hashing with Output Embeddings**

3.1 Introduction

The presence of social networks, image hosting websites like Imgur and Flickr, and the ubiquity of mobile phones with cameras have resulted in large-scale proliferation of images on the internet. Given a database of images, image retrieval seeks to return images from the database that are most similar to a query. Recent applications for content based image retrieval (CBIR) include Google's "Search by Image" feature [35] and TinEye [36], which enable users to search for an image using a closely related image, rather than a keyword. Performing image retrieval on databases with billions of images is challenging due to the linear time complexity of nearest neighbor retrieval algorithms. Image hashing [7,8,11,12,15,37] addresses this problem by obtaining similarity preserving binary codes which represent high dimensional floating point image descriptors and offer efficient storage and scalable retrieval with sub-linear search times. These binary hash-codes can be learned in unsupervised or supervised settings. Unsupervised hashing algorithms map nearby points in a metric space to similar binary codes. Supervised hashing algorithms try to preserve semantic label information in the Hamming space. Images that belong to the same class are mapped to similar binary codes.

In this work, we develop a new approach to supervised hashing, which we motivate with the example shown in Figure 3.1. Consider an image retrieval problem that involves a database of animals and a query image of a leopard. Now consider the following three scenarios :

1. If the retrieval algorithm returns images of leopards, we can deem the result to be absolutely satisfactory.

2. If the retrieval algorithm returns images of dolphins, whales or sharks, we consider the results to be absolutely unsatisfactory because not only are dolphins not leopards, they do not look anything at all like leopards.

3. If the retrieval algorithm returns the image of a jaguar or a tiger, we would be reasonably satisfied with the results. Although a jaguar is not the same as a leopard, it is semantically similar to one.

In this example, leopards, dolphins, whales, sharks, jaguars and tigers all belong to different categories. However, some of these categories are more closely related to each other than to other categories. Animals which fall under the "big cat" (*Panthera*) genus are related to each other, as are the large aquatic vertebrates like dolphins, sharks and whales. We designate the related categories as "siblings". The question now becomes : how can we construct hash functions that rank sibling class images ahead of unrelated class images? Traditional supervised hashing algorithms like [7,8] use discrete binary labels to compute similarity between classes. We, on the other hand, consider information outside of the image domain to define inter-class relationships.

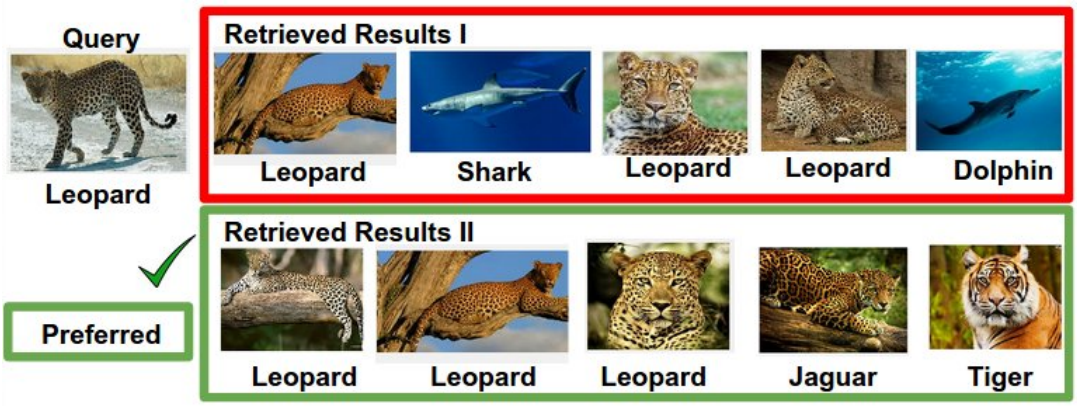

Figure 3.1: We prefer results II over I because they tend to retrieve images of classes(jaguar and tiger) related to the class label of the query(leopard), rather than unrelated classes(sharks and dolphins).

To study the relationships between categories, Weinberger *et al.* [38] suggested the concept of "output embeddings" - vector representations of category information in Euclidean space. There has been extensive work on "input embeddings", which are vector-space representations of images [39–41], but less work has been done on output embeddings, which map similar category labels to similar vectors in Euclidean space. In an output embedding space of animals, we would expect to have embeddings for labels so that chimpanzees, orangutans and gorillas are near each other as are leopards, cheetas, tigers and jaguars. In Section 3.2, we describe various methods from the literature to obtain output embeddings.

Our method, which uses output embeddings to construct hash codes in a supervised framework is called SHOE: Sibling Hashing with Output Embeddings (Figure 3.2). Our motivation for doing this is the following : it is our belief that in the case of supervised hashing, a more desirable algorithm will retrieve images of

sibling classes ahead of images of unrelated classes, while maintaining the retrieval performance for images of the same class. This improves the overall visual quality of retrieved results. We perform extensive retrieval experiments on the Caltech-UCSD Birds(CUB) dataset [42], SUN Attribute Dataset [43] and ImageNet [44]. Our hash-codes can also be used to do classification, and we report accuracy on the CUB dataset using a nearest-neighbor classifier which is better than R-CNN and its variants [45, 46].

The contributions of our work are as follows :

1. To the best of our knowledge, our approach is the first to introduce the problem of learning supervised hash functions using the modality of output embeddings.

2. We propose a joint learning method to solve the above problem, and perform retrieval and classification experiments to experimentally validate our method.

3. We propose two new evaluation criteria - "sibling metrics" and "weighted sibling metrics", for gauging the efficacy of our method.

4. We significantly boost retrieval and classification performance by applying Canonical Correlation Analysis [47] on input features, and learn hash functions using output embeddings on these features.

The remainder of this work is arranged as follows. Section 3.2 describes related work. In Section 3.3 we describe our hashing framework, and carry out experiments in Sections 3.4 and 3.5. We conclude in Section 3.6.

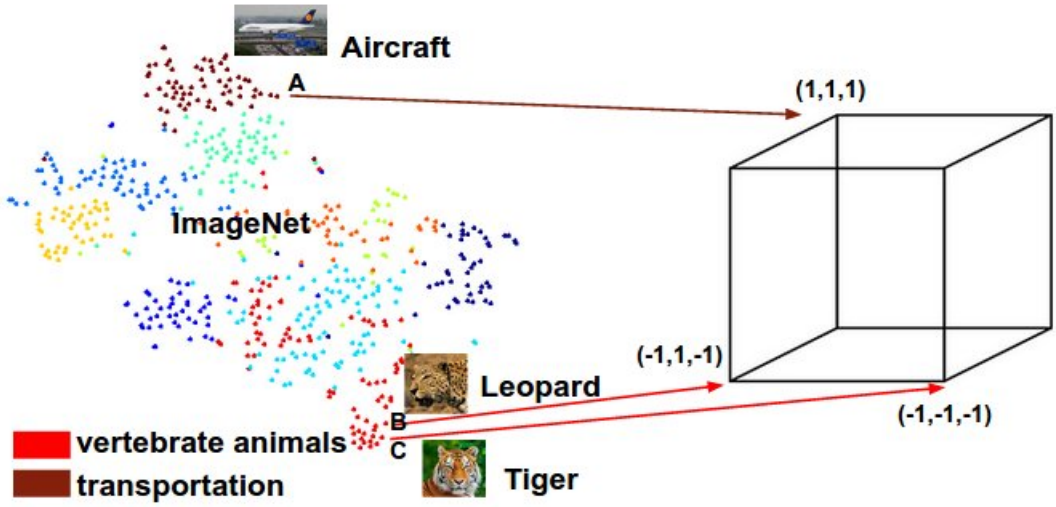

Figure 3.2: We perform k-means clustering on Word2Vec embeddings [1] of ImageNet classes. The principle behind SHOE is that images belonging to related classes (like leopard or tiger, which are nearby in the output embedding space) are mapped to nearby binary codes (represented by points on a binary hypercube). Images belonging to unrelated classes (like leopards and aircraft) are mapped to distant binary codes. This figure was created using [2] and is for illustrative purposes.

3.2 Related Work

Work on image hashing can be divided into unsupervised and supervised methods. For the purpose of brevity, we only consider supervised methods. Supervised Hashing algorithms are based on the objective function of minimizing the difference between hamming distances and similarity of pairs of data points. Supervised Hashing with Kernels(KSH) [7] uses class labels to determine the similarity. Points are considered 'similar' (value '1') if they belong to the same class and 'dissimilar'

(value '-1') otherwise. They utilize a simplified objective function using the relation between the hamming distance and inner products of the binary codes. A sequential greedy optimization is adapted to obtain supervised projections. FastHash [8] also uses a KSH objective function but employs decision trees as hash functions and utilizes a GraphCut based method for binary code inference. Minimal loss hashing [15] uses a structured SVM framework [48] to generate binary codes with an online learning algorithm.

All these methods except KSH categorize the input pairs to be either similar or dissimilar. KSH allows a similarity 0 for related pairs, but the authors only use it to define metric neighbors and not for semantic neighbors. FastHash entirely ignores the related pairs, as it weighs the KSH loss function by the absolute value of the label. Furthermore, their work does not support a similarity value other than 1 and -1, as it violates the submodularity property - a crucial property required to solve the problem in parts. The idea of ordering binary codes has been recently proposed in Zhang *et al.* [49], however, this has been done in an unsupervised setting. To the best of our knowledge, we are the first to use similarity information of sibling classes in a supervised hashing framework to obtain binary codes that respect relation between classes. Our work is also different from others as we compute similarity from output embeddings and use a joint learning framework to learn the sibling similarity. We now discuss related work on output embeddings.

Output embeddings can be defined as vector representations of class labels. We use these vector representations to compute the distance between categories in the label space, in the belief that sibling classes are nearby in this space compared

to unrelated ones. Output embeddings can be thought of as a modality exclusive of the pixel domain (or "input embeddings"), and can be divided into two types: data-independent and data-dependent. Some of the data-independent embeddings include [42,43,50–52]. Langford *et al.* [50] constructed output embeddings randomly from the rows of a Hadamard matrix where each embedding was a random vector of 1 or -1. Data dependent embeddings, on the other hand, can be constructed from side information about classes such as attributes, or a Linnean hierarchy, and are available with datasets such as [42, 43, 51]. In WSABIE [53], the authors jointly learn the input embeddings and the output embeddings to maximize classification accuracy in a structural SVM setting. Akata *et al.* [54] uses the WSABIE framework to learn fine grained classification models by mapping the output embeddings to attributes, taxonomies and their combination.

The binary hash-codes that SHOE learns on the CUB and SUN datasets use attributes as output embeddings, just like [54]. These attributes are provided either by expert oracles or Amazon Mechanical Turk - hence, output embeddings can be considered as a separate modality of information. For ILSVRC2010 experiments, we use a taxonomy derived embedding similar to [52]. In a taxonomy embedding, a binary output embedding vector is obtained, where each node in the class hierarchy and its ancestors are represented as 1 while non-ancestors are represented as 0. Deng *et al.* [55] show that classification that takes hierarchies into account can be informative. Mikolov *et al.* [1] use a skip-gram architecture trained on a large text corpus to learn output embeddings for words and short phrases. These Word2Vec embeddings are used by [56] for large scale image classification and zero-shot learning. Finally,

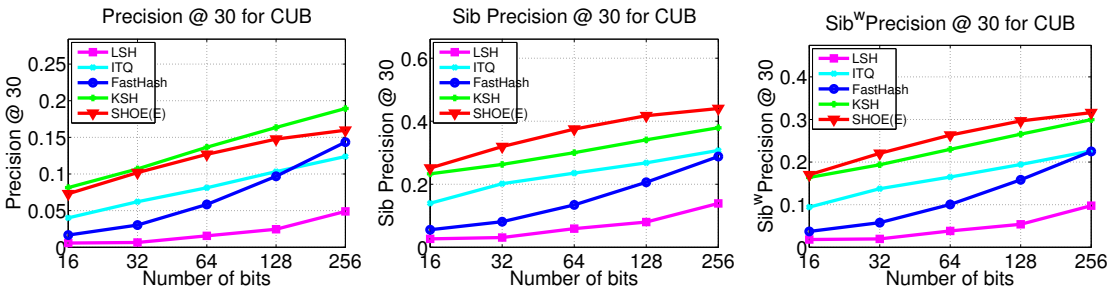

Figure 3.3: Retrieval on CUB dataset comparing our method SHOE(E) with the state-of-the art hashing techniques. The above plots report precision, sibling precision and weighted sibling precision for top 5 sibling classes for bits $c = \{16, 32, 64, 128, 256\}$.

output embeddings can even be learned from the data. For example, [57] exploits co-occurences of visual concepts to learn classifiers for unseen labels using known classifiers. All these methods use output embeddings for classification and zero shot learning, but none have used them to learn binary codes for retrieval.

3.3 Method

3.3.1 Preliminaries

Given a training set $M = \{(x_1, y_1), ..., (x_N, y_N)\}$ of N (image,label) pairs with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$, let $\phi : \mathcal{X} \rightarrow \bar{\mathcal{X}} \in \mathcal{R}^d$ be the input embedding function and $\psi : \mathcal{Y} \rightarrow \bar{\mathcal{Y}} \in \mathcal{R}^e$ be the output embedding function. We wish to learn binary codes b_i, b_j of length c (i.e., $b_i \in \{-1, 1\}^c$) such that for pairs of training images, the Hamming distance between the codes preserve the distance between their class

Method	mAP	Sib_{mAP}	Sib^{w}_{mAP}
SHOE(E)	0.111	**0.250**	**0.174**
KSH	**0.113**	0.133	0.108
FastHash	0.045	0.062	0.047
ITQ	0.060	0.119	0.084
LSH	0.013	0.044	0.028

Table 3.1: Retrieval on CUB dataset comparing our method SHOE(E) with the state-of-the art hashing techniques. The table reports mAP, Sibling and Weighted Sibling mAP for $c = 64$ bits.

labels (given by their corresponding output embedding vectors). In other words, for a given query image, retrieved results of sibling(unrelated) classes ought to be ranked higher(lower). To this end, we obtain the following objective function:

$$\min_{b} \sum_{i=1}^{N} \sum_{j=1}^{N} (\frac{1}{c} d_H(b_i, b_j) - d_E(\psi(\bar{y}_i), \psi(\bar{y}_j)))^2. \tag{3.1}$$

where $d_H(b_i, b_j)$ is the Hamming distance between binary codes b_i and b_j and d_E is the Euclidean distance between the normalized output embedding vectors $\psi(\bar{y}_i)$ and $\psi(\bar{y}_j)$.

For an input image x with input embedding vector $\phi(x)$, we obtain binary code b of length c bits. Each bit is computed using a hash function $h_l(x)$ that takes the form :

$$h_l(x) = sgn(w_l\phi(x)), w_l \in \mathcal{R}^d. \tag{3.2}$$

To learn c such hash functions $\mathcal{H} = \{h_l | l = 1, \ldots, c\}$, we learn c projection vectors $W = [w_1, w_2 \ldots, w_c]$, which we compactly write as $H(x) = sgn(W\phi(x))$, $W \in \mathcal{R}^{c \times d}$. Without loss of generality, we can assume that $\phi(x)$ is a mean-centered feature. This ensures we obtain compact codes by satisfying the balanced property of hashing (i.e, each bit fires approximately 50% of the time) [12]. $\phi(x)$ can be an input embedding that maps images to features in either kernelized or unkernelized forms.

Solving the optimization problem in Equation (1) is not straightforward, so we utilize the relation between inner product of binary codes and Hamming distances [7, 58], given as $2d_H(b_i, b_j) = c - b_i^T b_j$, where $b_i^T b_j = \sum_{l=1}^{c} h_l(\phi(x_i))\ h_l(\phi(x_j))$ is the inner product of the binary codes b_i and b_j. Note that the inner product of binary codes lies between $-c$ and $+c$, while the Hamming distance ranges from 0 to c, where the distance between the nearest neighbors is 0 and between the farthest neighbors is c . By unit normalizing the output embedding vectors, $\|\bar{\psi}(y)\| = 1$, we exploit the relationship between Euclidean distances and the dot products of normalized vectors, given as $d_E(\bar{\psi}(y_i), \bar{\psi}(y_j)))^2 = 2 - 2\bar{\psi}(y_i)^T\bar{\psi}(y_j)$, and obtain the following objective function:

$$\min_{\mathcal{H}} \sum_{i=1}^{N}\sum_{j=1}^{N}(\frac{1}{c}\sum_{l=1}^{c} h_l(\phi(x_i))h_l(\phi(x_j)) - \bar{\psi}(y_i)^T\bar{\psi}(y_j))^2 \tag{3.3}$$

Let $o_{ij} = \bar{\psi}(y_i)^T\bar{\psi}(y_j)$ and as a consequence of the unit normalization of $\bar{\psi}(y_i)$, $-1 \leq o_{ij} \leq 1$, which implies that the similarity between same classes is **1** and

similarity between different classes is as low as **-1**. The objective ensures that the learned binary codes preserve the similarity between output embeddings, which is required for supervised hashing and our goal of ranking related neighbors before farthest neighbors.

This is similar to the KSH [7] objective function, except that KSH assumes that o_{ij} takes only values 1 (-1) for similar (dissimilar) pairs defined with semantic information. Their work also accomodates the definition of the $o_{ij} = 0$ for related pairs but only for metric neighbors. Our work is different from theirs, as we emphasize the learning of binary codes that preserve the similarity between the classes, given by o_{ij}. Regardless of the definition of o_{ij}, our optimization is similar, and so we employ a similar sequential greedy optimization for minimizing (3.3). We refer the reader to Section 3.3.5 and [7] for further details.

3.3.2 Evaluation Criteria

Standard metrics like precision, recall and mAP defined for semantic neighbors are not sufficient to evaluate the retrieval of the sibling class images. To measure this, we define *sibling precision*, *sibling recall* and *sibling average precision* metrics. Let $R_y : (y_i, y_j) \rightarrow rank$ return the rank of class y_j for a query class y_i, $0 \leq rank \leq L$, where L is the number of classes. Note that, $R_y(y, y) = 0$ for the same class. The ranking R_y is computed by sorting the distance between the output embedding vectors $\psi(y)$ and $\psi(y^*)$, $y^* \in \mathcal{Y} \setminus y$. We obtain the weight of the sibling class used for evaluation using the following functions for Sibling(Sib_m) and Weighted

Sibling(Sib_m^w) metrics:

$$Sib_m : (y_i, y_j, R_y) \rightarrow \mathbb{I}(R_y(y_i, y_j)) \leq m)$$

$$Sib_m^w : (y_i, y_j, R_y) \rightarrow \frac{m - R_y(y_i, y_j)}{m} * \mathbb{I}(rank \leq m)$$

where, m is the number of related classes for each query class and $\mathbb{I}(.)$ is the Indicator function that returns 0 when $rank > m$. The Sibling and Weighted Sibling precision@k, recall@k and mAP is defined as:

$$s^w_{precision_q}@k = \frac{\sum_{l=1}^{k} Sib_m^w(y_q, y_{q_l}, R_y)}{k} \tag{3.4}$$

$$s^w_{recall_q}@k = \frac{\sum_{l=1}^{k} Sib_m^w(y_q, y_{q_l}, R_y)}{\sum_{p=1}^{L} \mathcal{N}_l * \mathbb{I}(Sib_m^w(y_q, y_p, R))} \tag{3.5}$$

$$s^w_{AP_q} = \sum_{k=1}^{N} s_{precision_q}@k \times \triangle s_{recall_q}@k \tag{3.6}$$

$$s^w_{mAP} = \frac{\sum_{q=1}^{Q} s_{AP_q}}{Q} \tag{3.7}$$

In the above equations, y_{q_l} refers to the class of the l^{th} retrieved image.

3.3.3 Preliminary Experiments

We evaluate the proposed hashing scheme that takes into account structure of the related classes with the following performance metrics: Precision@k, mAP and their sibling versions previously defined.

- ***Datasets:*** To test our approach, we use datasets that contain information about class structure. The CUB-2011 dataset [42] contains 200 fine-grained bird categories with 312 attributes and is well suited for our purpose. In this dataset, although both binary and continuous real-valued attributes are available, we use only the mean-centered real-valued attributes as output embeddings $\psi(y)$. We obtain ranking R_y for each class y based on these attribute embeddings. There are 5994 train and 5774 test images in the dataset. We select a subset of the dataset of size 2000 for training, use the whole train set for retrieval and all test images as queries. Ground truth for a query is defined label-wise and each query class has approximately 30 same class neighbors in the retrieval set. For input embeddings, we extract state-of-the-art 4096 dimesional Convolution Neural Network (CNN) features from the fc7 layer for each image using the Caffe Deep Learning library developed by [33]. We kernelize the CNN features which take the form :$\sum_{i=1}^{p} \kappa(x, x_i)$ where κ is a radial basis kernel, and p is the cardinality of a subset of training sample, designated as "anchor points". We refer to these as CNN+K features and they are inherently mean-centered.

- ***Comparison methods:*** We compare our method with raw output embeddings:SHOE(E), with the following supervised and unsupervised hashing schemes: KSH [7], FastHash [8], ITQ [28] and LSH [11]. We use their publicly available implementations and set the parameters to obtain the best performance. It is important to note that none of these methods utilize the distri-

bution of class labels in the output embedding space. The closest comparison would be to use KSH, setting the similarity of semantic class neighbors to 0 value. For this purpose, we obtain top-m related pairs for each class using R_y and set the similarity to 0 for related pairs. To evaluate the unsupervised ITQ and LSH, we zero-center the data and apply PCA to learn the projections. We use CNN+K features with $p = 300$ for evaluating SHOE(E) and KSH since we learn linear projections in these methods, unlike FastHash which learns non-linear decision trees on linear features.

- ***Results:*** Figure 3.3 shows the precision@30, sibling and weighted sibling precision@30 plots for 5 related classes, i.e. $m = 6$(+1 for the same class) by encoding the input embeddings to bits of length c={16, 32, 64, 128, 256}. Table 3.1 shows the recall and mAP and its sibling variants for 64 bits. We observe that SHOE(E) does better than baselines for both sibling and weighted sibling precision metrics for top-30 retrieved neigbhors for all bit lengths, but there is a loss in precision compared to the KSH method. In their work, FastHash [8] shows better performance compared to KSH. However, it does not perform well here because of the large number of classes and few training samples available per class.

3.3.4 Analysis

Experiments in Section 3.3.3 show that using similarity directly from output embeddings actually reduces the performance for same classes, while improving it for

sibling classes. To analyse this, we obtained top-m related classes using ranking R_y and assigned a constant $o_{ij} = \theta$ value (previously, in Equation(3.3), we had defined $o_{ij} = \bar{\psi}(y_i)^T\bar{\psi}(y_j)$). θ measures the similarity between a class and a related class. Figure 3.4 show the performance of our method(SHOE) for varying θ values for 64 bits on CUB-2011 dataset. Results reveal that, when a fixed similarity is used, we actually gain performance from the sibling class training examples, and this gain is maximized for negative values of θ, i.e $-1 < \theta < 0$. The intuition behind this is: when θ is close to 1, the learned hash-code would not discriminate well enough between identical classes and sibling classes. For instance, in our "database of animals" example, we would learn hash-codes that nearly equate leopards with jaguars, which is not what we desire. On the other hand, when θ is close to -1, a hash-code for a leopard image will be learned mostly from other training images of leopards, but with slight consideration towards training images of its sibling classes. When θ is assigned -1, the sibling classes aren't considered at all, so our method becomes identical to KSH [7]. We are now interested in learning θ simultaneously with the hash functions during the training phase.

3.3.5 SHOE Revisited

We observe that the objective function that we want to minimize in Equation (3.3) can be split into three parts - for identical classes, sibling classes and unrelated classes, respectively. We also observe from the preceding analysis that precision and recall metrics improve for negative values of θ. Therefore, we add reg-

Method	CUB-2011(2000 train: (Pre,re)@100)						SUN Attribute(3585 train: (Pre,re)@25)					
	$pre\|re$		$Sib_{pre\|re}$		$Sib^{w}_{pre\|re}$		$pre\|re$		$Sib_{pre\|re}$		$Sib^{w}_{pre\|re}$	
SHOE(L)	**0.108**	**0.180**	**0.325**	**0.091**	**0.229**	**0.109**	**0.042**	**0.105**	**0.140**	**0.058**	**0.094**	**0.067**
SHOE(E)	0.077	0.128	0.285	0.079	0.190	0.090	0.023	0.056	0.095	0.040	0.060	0.043
KSH	0.089	0.149	0.225	0.062	0.165	0.079	0.035	0.087	0.091	0.038	0.066	0.047
ITQ	0.069	0.115	0.214	0.060	0.147	0.070	0.031	0.078	0.097	0.040	0.066	0.047
FastHash	0.035	0.117	0.097	0.054	0.069	0.065	0.006	0.015	0.019	0.008	0.013	0.009
LSH	0.013	0.042	0.053	0.030	0.034	0.032	0.005	0.012	0.021	0.009	0.013	0.009
	with CCA features											
SHOE(L)+CCA	**0.142**	**0.474**	**0.437**	**0.243**	**0.308**	**0.293**	**0.060**	**0.150**	**0.197**	**0.082**	**0.135**	**0.096**
SHOE(E)+CCA	0.121	0.402	0.416	0.232	0.284	0.270	0.040	0.099	0.150	0.062	0.098	0.070
KSH+CCA	0.136	0.454	0.322	0.179	0.244	0.232	0.057	0.143	0.138	0.058	0.102	0.073
ITQ+CCA	0.095	0.318	0.203	0.113	0.158	0.150	0.029	0.071	0.067	0.028	0.050	0.035

Table 3.2: Comparing Precision, recall and their sibling variants with our methods(SHOE(L) and SHOE(E)) and several baselines for 64 bits. We here show results with and without applying CCA projections for all the methods except FastHash and LSH as their performance decreases. We see that SHOE performs significantly better than the baselines for sibling metrics and performs as well as the baselines for standard metrics.

ularizer term $\lambda \|\theta + 1\|^2$ to the objective function, which becomes small when θ lies close to -1. For easier notation, we denote $h_{l_i} = h_l(\phi(x_i))$. Our modified objective function now becomes :

$$\min_{W,\theta} \sum_{i,j\in}^{N} \sum_{\substack{same\\class}}^{N} (\frac{1}{c}\sum_{l=1}^{c} h_{l_i} h_{l_j} - 1)^2 + \sum_{i,j\in}^{N} \sum_{\substack{sibling\\class}}^{N} (\frac{1}{c}\sum_{l=1}^{c} h_{l_i} h_{l_j} - \theta)^2$$
$$+ \sum_{i,j\in}^{N} \sum_{\substack{unrelated\\class}}^{N} (\frac{1}{c}\sum_{l=1}^{c} h_{l_i} h_{l_j} + 1)^2 + \lambda \left\|\theta + 1\right\|^2 \tag{3.8}$$

Let $\mathcal{H}_{ij_c} = \frac{1}{c}\sum_{l=1}^{c} h_{l_i} h_{l_j}$ denote the sum of the inner product of the binary codes b_i and b_j of length c. We now compute the derivative of Equation (3.8) w.r.t θ, set it to 0 and solve for each θ. We obtain :

$$\theta = \theta_c = \frac{\sum_{i,j\in}^{N} \sum_{\substack{sibling\\class}}^{N} (\mathcal{H}_{ij_c} - \lambda)}{c(n_{sib} + \lambda)} \tag{3.9}$$

where n_{sib} is the number of sibling pairs in the training data. We have thus obtained a closed form solution for the optimal θ. However, we cannot calculate θ directly as we do not learn all the bits at once. Therefore, we employ a two step alternate optimization procedure that first learns the bits and then an approximate θ_l value calculated from the previously learned bits. For the first iteration, we use an initial θ_0 value, computed from the similarity of the output embeddings. The two step optimization procedure for learning the l^{th} hash function is:

1. Step 1 : We optimize for Equation (3.3), keeping θ_{l-1} constant and updating the projection vector W, thus learning hash-code bits $h_l(\phi(x_i))$.

2. Step 2 : We keep the hash-code bits $h_l(\phi(x_i))$ constant and learn θ_l for each image pair using Equation (3.9).

Let $\mathcal{O}$ denote the inner product of output embedding vectors such that o_{ij} is assigned 1 for same class pairs, θ for sibling class pairs and -1 for unrelated class pairs and $\mathcal{H}_l$ denote the l^{th} bit for all training samples. Now, we can write equation (3.8) as:

$$\min_{W,\theta} \left\| \frac{1}{c} \sum_{l=1}^{c} \mathcal{H}_l \mathcal{H}_l^T - \mathcal{O} \right\|_F^2 + \lambda \left\| \theta + 1 \right\|^2 \tag{3.10}$$

Utilizing the independence constraints [12, 59] for learning hashing functions, we rewrite the objective function to greedily solve for one bit at a time like in [7], given as:

$$\min_{W_l,\theta} \left\| \frac{1}{c} \mathcal{H}_l \mathcal{H}_l^T - (\mathcal{O} - \sum_{m=1}^{l-1} \mathcal{H}_m \mathcal{H}_m^T) \right\|_F^2 + \lambda \left\| \theta + 1 \right\|^2 \tag{3.11}$$

Let $\mathcal{M}_{l-1} = \mathcal{O} - \sum_{m=1}^{l-1} \mathcal{H}_m \mathcal{H}_m^T$. Replacing $\mathcal{H}_l$ with $sgn(W_l \phi(X))$ in equation (3.11) and expanding, we get:

$$\min_{W_l,\theta} -sgn(W_l \phi(X))^T \mathcal{M}_{l-1} sgn(W_l \phi(X)) + \lambda \left\| \theta + 1 \right\|^2 \tag{3.12}$$

As discussed in the above two step procedure, we solve first for W_l and then for θ. In the first step, we relax the sgn function and scale the projected vectors to obtain initial solution W_l, which is the solution of the generalized eigenvalue problem $\phi(X)^T \mathcal{M}_{l-1} \phi(X) W_l = \sigma \phi(X)^T \phi(X) W_l$. Then, we approximate the sgn function with $tanh$. The gradient for the above equation w.r.t to W_l is now given as:

$$\delta g = -\phi(X)^T ((\mathcal{M}_{l-1} b) \circ (1 - b \circ b)) \tag{3.13}$$

Given the gradient, we solve the optimization using a fast iterative procedure like Nesterov's gradient descent [60] or FASTA [61]. In the second step, we learn θ as

given in Equation (3.9). We then update $\mathcal{M}_{l-1}$ with the new value of θ and repeat the optimization procedure for the next bit until we learn the required number of bits.

3.3.6 Supervised Dimensionality Reduction

As our datasets contain class label information and corresponding output embeddings, we have explored the idea of supervised dimensionality reduction for input embeddings $\phi(x) \in R^d$ to $\omega(x) \in R^c (c \ll d)$, given the output embeddings $\psi(y) \in R^E$. There are many supervised dimensionality reduction techniques available in the literature like Canonical Correlation Analysis (CCA) [47] and Partial Least Square Regressions [62], for example. In particular, we have used CCA [47] ($\phi \rightarrow \omega$) to extract a common latent space from two views that maximizes the correlation with each other. [28] also leveraged the label information using CCA to obtain supervised features prior to binary encoding. However, they limit their output embeddings to take the form of one vs remainder embeddings: : $\psi(y) \in \{0,1\}^L$ is a L-dimensional binary vector with exactly one bit set to 1 i.e. $\psi(y)_y = 1$ where L is the number of class labels. On the other hand, we apply CCA to the general form of output embeddings that are real valued continuous attributes capturing structure between the classes. We observe that when supervised features with CCA-projections are used, we obtain a significant boost in performance ($\approx 100\%$ improvement) for all of our evaluation metrics.

Method	CUB-2011(5000 train)						SUN Attribute(7000 train)					
	$pre\|mAP$		$Sib_{pre\|mAP}$		$Sib^{w}_{pre\|mAP}$		$pre\|mAP$		$Sib_{pre\|mAP}$		$Sib^{w}_{pre\|mAP}$	
	@100	mAP	@100	mAP	@100	mAP	@25	mAP	@25	mAP	@25	mAP
SHOE(L)+CCA	**0.211**	**0.527**	0.561	0.467	**0.417**	0.416	**0.114**	0.201	**0.311**	**0.239**	**0.225**	**0.193**
SHOE(E)+CCA	0.183	0.429	**0.575**	**0.533**	0.411	**0.429**	0.078	0.134	0.240	0.205	0.167	0.152
KSH+CCA	0.204	**0.526**	0.421	0.290	0.335	0.303	**0.115**	**0.220**	0.227	0.130	0.178	0.126
ITQ+CCA	0.109	0.256	0.193	0.125	0.157	0.129	0.039	0.070	0.081	0.044	0.062	0.041
FastHash	0.109	0.246	0.192	0.120	0.156	0.124	0.014	0.021	0.029	0.017	0.022	0.014
LSH	0.018	0.017	0.069	0.049	0.045	0.032	0.008	0.009	0.030	0.018	0.019	0.012

Table 3.3: Comparing Precision, mAP and their sibling variants with our methods(SHOE(L) and SHOE(E)) and several baselines for 128 bits. We apply CCA projections for all the methods except FastHash and LSH as their performance decreases. For sibling metrics, SHOE performs significantly better than the baselines and performs as well as the baselines for standard metrics. We use 1000 anchor points for CUB dataset and 1434 anchor points for SUN attribute dataset.

3.4 Experiments

We evaluate our method on the following datasets: Caltech-UCSD Birds (CUB) Dataset [42], the SUN Attribute Dataset [43] and Imagenet ILSVRC2010 dataset [44]. We extract CNN features from [33], as mentioned in Section 3.3.3. In the case of CUB dataset, we extract CNN features for the bounding boxes that accompany the images. For CUB and SUN datasets, we create two training sets of different size to examine the variation in performance with number of training examples. For all the datasets, we define the ground truth using class labels.

Datasets: We have described the CUB dataset in Section 3. The ImageNet ILSVRC 2010 dataset is a subset of ImageNet and contains about 1.2 million images distributed amongst 1000 classes. We uniformly select 2 images per class as a test set and use the rest as retrieval set. We select 5000 training and $p = 3000$ anchor point images by uniformly sampling across all classes. We obtain the output embeddings for the ImageNet class using the method of Tsochantaridis [52]. Each of the 74401 synsets in ImageNet is a node in a hierarchy graph and using this graph, we obtain the ancestors for each class in ILSVRC 2010 dataset. We then construct a matrix $O_{Imagenet} = \{0, 1\}_{1000\times74401}$, where the j^{th} column of the i^{th} row is set to 1 if the j^{th} class is an ancestor of the i^{th} class, 0 otherwise. Thus, the output embedding of each class is represented by a row of $O_{Imagenet}$.

The SUN Attribute dataset [43] contains 14340 images equally distributed amongst 717 classes, accompanied by annotations of 102 real valued attributes. We partition the dataset into equal retrieval and test sets, each containing 7170 images.

We derive two variants from the retrieval set - the first has 3585(5 per class) training and 1434(2 per class) anchor point images, while the second has 7000(10 per class) training and 3000(4 per class) anchor point images. In this dataset, each test query has 10 same class neighbors and 50 sibling class neighbors in the retrieval set. We compute a per class embedding by averaging embedding vectors for each image in the class.

Evaluation Protocol: For binary hash-codes of length $c = \{16, 32, 64, 128, 256\}$, we evaluate SHOE using standard, sibling and weighted sibling flavors of precision, recall and mAP. Two variants of SHOE are used - SHOE(E), which uses raw output embeddings, and SHOE(L), which is learned using the method in Section 3.3.5. For the smaller CUB and SUN subsets, we compute mAP and their sibling versions. For the big subsets, in addition to mAP, we also compute precision. In the case of ImageNet, we compute precision@50, recall@10K, mAP and their sibling variants.

Results: We compare our work to state-of-the-art methods in image-hashing literature mentioned in Section 3.3.3 and present the results in Figure 3.5, Tables 3.2 and 3.3. Figure 3.5 and Table 3.2 represent experiments performed on a smaller dataset with 2000 train, 1000 anchor points for CUB and 3585 train, 1434 anchor point images for SUN Attribute dataset. Table 3.3 shows experiments that use 5000 train, 3000 anchor for CUB and 7000 train, 3000 anchor points for SUN Attribute. For the baselines, we use publicly available implementations and set the parameters to obtain the best performance. We use CNN+K features as input embeddings for SHOE and KSH.

In Figure 3.5, the rows represent weighted sibling, sibling and mAP metrics.

The first two columns represent experiments without and with CCA projections on the CUB dataset, while the latter two columns represent the same on the SUN Attribute dataset. We notice that without CCA projections, we outperform the baselines on standard metrics, and comfortably outperform on sibling metrics. These results confirm our expectation that with fewer training samples, learning hash codes with SHOE benefits from training samples of related classes. With output embedded CCA projections, our method performs as well as the best baseline on standard metrics, and comfortably outperforms on sibling metrics. CCA projections significantly improve the performance for SHOE, KSH and ITQ, while it lowers the performance for FastHash and LSH.

Table 3.2 does the same comparison with 64 bits and fewer training samples. Once again, we notice that our method with and without CCA, SHOE(L) beats all baselines on both standard and sibling metrics, while SHOE(E), which uses raw output embeddings, without the extra optimization step in Section 3.3.5, outperforms comfortably on sibling metrics, but does worse on the standard metrics. Table 3.3 shows similar trends for 128 bits for larger training sets, but we notice that the gap in performance on standard metrics between SHOE and KSH reduces when CCA projections are applied, but not on the sibling metrics. We think this is because the CCA step utilizes output embeddings and hence, improves performance of both KSH and SHOE. Finally, we conduct experiments on the Imagenet ILSVRC 2010 dataset, which contains 1000 classes. Results from Figure 3.6 show that with CCA features, SHOE surpasses KSH on recall@10K and sibling precision metrics, while performing as well as KSH on precision@50 and mAP. Qualitative results which

Method	pre	s_{pre}	s^w_{pre}
SHOE(L)+CCA	24.5	**32.6**	**29.1**
KSH+CCA	**24.8**	30.4	28.0
ITQ+CCA	6.5	10.8	9.9
Method	mAP	s_{mAP}	s^w_{mAP}
SHOE(L)+CCA	**0.039**	**0.021**	**0.022**
KSH+CCA	0.036	0.010	0.014
ITQ+CCA	0.005	0.001	0.002

Table 3.4: Retrieval on ILSVRC2010 dataset comparing SHOE with state-of-the art hashing techniques. We use $5K$ training samples and CNN+K+CCA as features for all the binary encoding schemes. Table reports precision@50, mAP and their sibling versions for 256 bits.

compares SHOE and KSH can be found in Figure 3.7.

3.5 Fine-grained Category Classification

In this section, we demonstrate the effectiveness of our proposed codes for fine-grained classification of bird categories in CUB-2011 dataset. We propose a simple nearest neighbor pooling classifier that classifies a given test image by assigning it to the most common label among the top-k retrieved images. Let $R_x(q, x) \rightarrow rank$ give the ranking of the images retrieved based on our binary codes. Thus, $rank$ is 1

for the nearest neigbhor and $rank$ is N for the farthest neighbor, where N is the size of the database. Given such ranking, with $\mathcal{M}_q$ denoting the top-k ranked neighbors of a new query q, we define the k-nn pooling classifier as:

$$class_{predict}(q) = \arg\max_{y} \sum_{x \in \mathcal{M}_q} \mathbb{I}(class(x) == y) \tag{3.14}$$

We use the above model to obtain the classification accuracy on the CUB dataset with 200 categories from top-10 neighbors. In particular, we obtain the following accuracies: top-1 accuracy(top-1) measures if the predicted class matches the ground truth class, top-5 accuracy(top-5) measures if one of the top-5 predicted classes match the ground truth class and sibling accuracy(sib) measures if the predicted class is one of the sibling classes of the ground truth class. As a baseline, we train a linear SVM model on the CNN features. We compare our proposed binary codes SHOE(L), KSH and state-of-the-art fine grained classification models that use CNN features. For this experiment, we use bounding box information, but do not use any part-based information available with the datasets. Hence, we do a fair comparison between methods with no part-based information. Table 3.5 shows the classification performance over the 5794 test images with approximately 30 images for each of the 200 categories.

Features: For each of the binary coding schemes(SHOE, KSH, ITQ), we use the CNN+K+CCA(kernelized CNN with CCA projections) features as input embeddings and mean-centered attributes as the output embeddings. For the experiments, we used only 128 bit codes, while CNN features are 4096 dimensional vectors.

Method	top-1	top-5	sib	Compression
Baseline(SVM)	50.6	75.6	70.19	1
SHOE(L)+CCA	**52.51**	**77.8**	**72.4**	**1024**
KSH+CCA	52.48	75.1	69.06	**1024**
ITQ+CCA	27.5	43.4	37.6	**1024**
R-CNN [45]	51.5	-	-	1
Part-RCNN [46]	52.38	-	-	1

Table 3.5: Comparing classification accuracies for CUB dataset. For top-1 and sibling accuracy, we used $k = 10$ neighbors. To obtain top-5 accuracy, we used $k = 50$ neighbors. For the binary coding schemes, we used only 5000 of the 5994 train images to obtain 128 bits, while the classification models are trained on the full set. '-' indicates that the information is not available in their work.

Results: We observe that not only do the proposed binary codes obtain a marginal improvement in performance over the complex classification models in [45] [46], but they also offer an astounding compression ratio of **1024**. Also, the training and testing times of binary coding schemes are significantly smaller than those with SVM classification models.

3.6 Conclusion

The key idea of our work is to learn binary codes that respect the relationship between classes. We utilize output embeddings to define the similarity between classes and obtain binary codes that preserve class order. To the best of our knowledge, ours is the first work to do so. We devised a method to learn class similarity jointly with the hash function, along with new metrics for their evaluation. It is our belief that this scheme improves overall visual quality of the retrieval system, and we have validated this experimentally with sibling metrics. Our method, called SHOE, achieves state-of-the art image retrieval results over multiple datasets for hash codes of varying lengths. Our other innovation was to utilize CCA to learn a projection of features with output embeddings, which resulted in significant gains in both retrieval and classification experiments. Upon applying this approach to all methods, we perform comparable to or better than all baselines over all datasets.

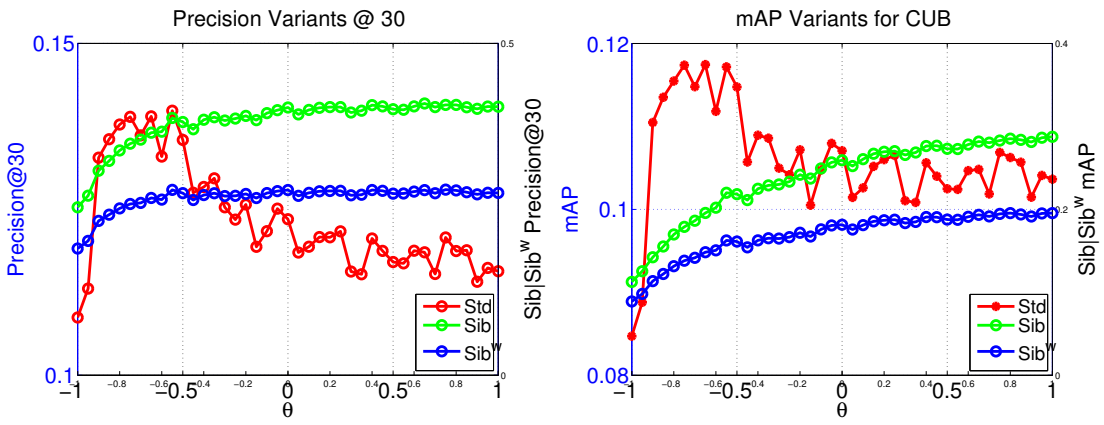

Figure 3.4: Retrieval on CUB dataset evaluating the performance of our method(SHOE) for varying θ values and $p = 1000$. The left and right y-axis show the standard metrics and sibling metrics respectively.

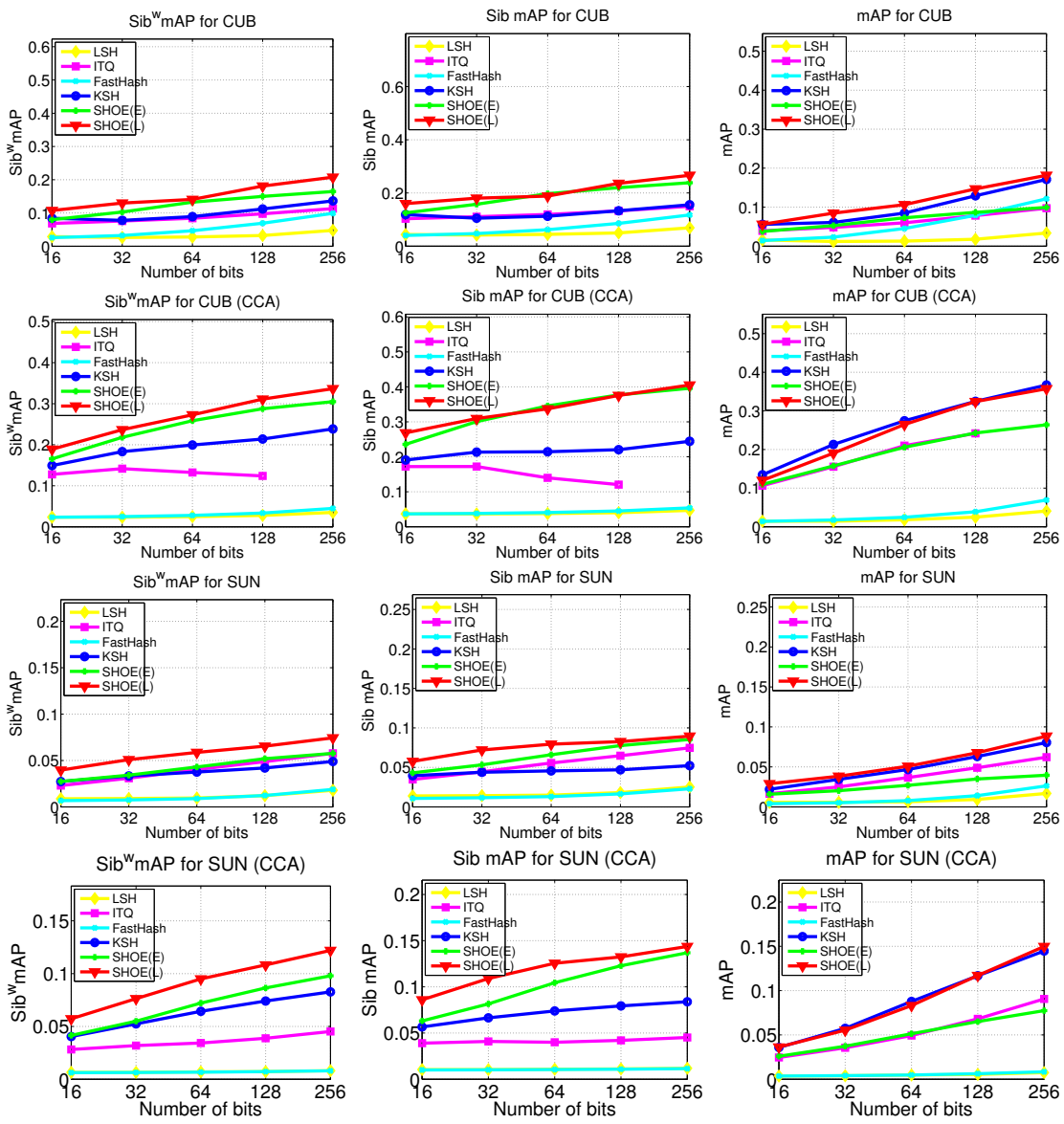

Figure 3.5: Retrieval on CUB-2011(first and second row) and SUN(third and fourth row) dataset comparing our methods SHOE(E) and SHOE(L) with the state-of-the art hashing techniques. The above plots report mAP, Sibling and Weighted Sibling mAP for top 5 sibling classes. For the CUB dataset, we used 2000 training samples and 1000 anchor points, while for the SUN attribute dataset, we used 3585 training samples and 1434 anchor points.

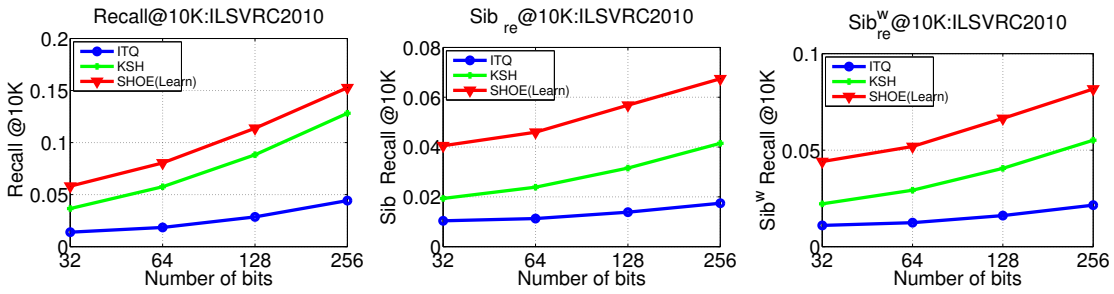

Figure 3.6: Retrieval on ILSVRC2010 dataset comparing SHOE with state-of-the art hashing techniques. We use $5K$ training samples and CNN+K+CCA as features for all the binary encoding schemes. The above plots report *recall*, Sib_{re}, Sib^{w}_{re} @10K for top 5 sibling classes for bits $c = \{32, 64, 128, 256\}$.

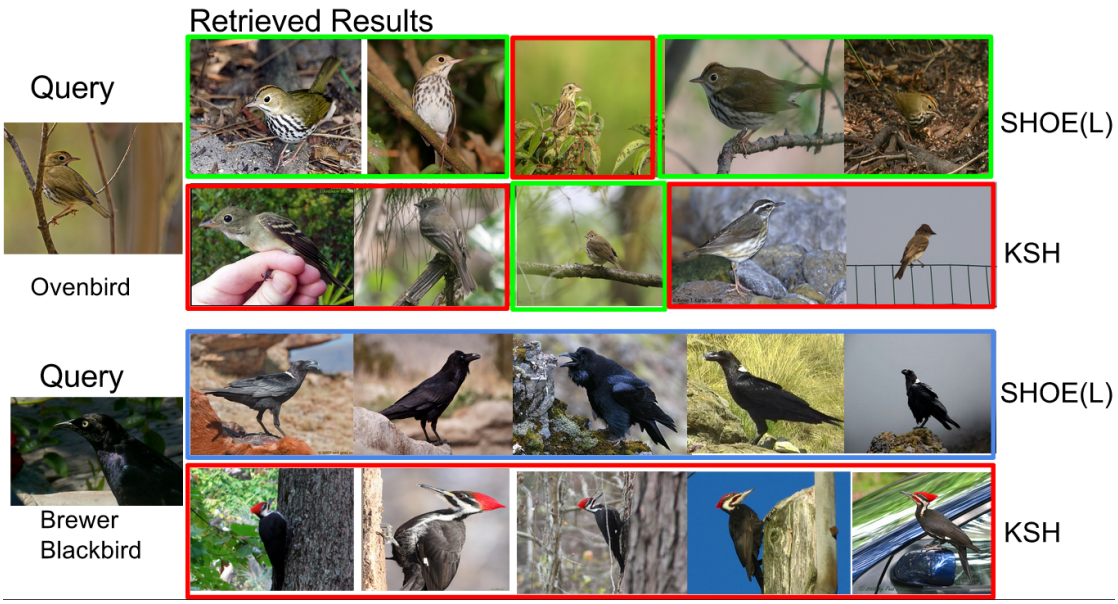

Figure 3.7: The first query is of an ovenbird. SHOE retrieves more ovenbirds than KSH. The second query is of a Brewer black-bird. Neither SHOE nor KSH retrieve Brewer black-birds. However, SHOE returns ravens, which are sibling classes of Brewer black-birds, whereas KSH retrieves pileated woodpeckers, which are unrelated to black-birds. Here, blue borders represent sibling classes.

Chapter 4: **Hierarchical Spherical Hashing for Compressing High Dimensional Vectors**

4.1 Overview

Large dimensional descriptors like Fisher Vectors (FV) [63, 64], Vector of Locally Aggregated Descriptors(VLAD) [65] and Locally Constrained Linear Codes(LLC) [66] have performed very well in problems like retrieval, classification and object detection [67–69]. A major bottleneck in using such vectors is their high storage requirements. For example, a typical Fisher Vector size ranges from 1000 to 0.5 Million with 16 to 4096 GMMs; that translates to 8 KB to 2 MB. The full dataset of ILSVRC2010 containing 1.4 Million images [44] needs 3 TBs of storage when storing FV's over 4096 GMMs. One can adapt dimensionality reduction techniques like PCA to reduce the storage requirements at the cost of losing structural properties of the original vectors. To preserve the discriminative power of such high-dimensional descriptors without losing structural properties, Perronnin et al [67, 68] found that they must be encoded using a large number of bits.

There have been many proposals for compressing high-dimensional descriptors to a small number of bits [11, 12, 28, 65, 70–72] by initially applying PCA, but less

has been done on compressing to a large number of bits. Perronin et al [67, 68] used sparsity constraints that employ thresholding based methods to encode to a large number of bits. Locality Sensitive Hashing [11] and Product Quantization [70] based methods can be applied to compress large descriptors to large numbers of bits at a storage cost. Specifically, to compress a 128K dimensional vector to 64K bits, LSH requires 32 GB for storing the projection matrix, which is a huge storage requirement. PQ encoding methods obtain excellent results for retrieval by partitioning the vectors and learning a codebook for each partition. Binary codes are obtained as a concatentation of the codebook indices to which each sub-vector belongs. Approximate nearest neighbors are then computed efficiently using symmetric/asymmetric distance between the query and the codebook learned for each partition, obtained by looking up the codebook indices. This method works best when an orthogonal rotation is applied to the data to balance variance, but incurs a huge storage cost for the rotation matrices. Gong et al [73] also proposed a fast bilinear projection method to compress large dimensional vectors without the storage overhead of projection matrices. They accomplish this by formulating the projection operation as the Kronecker product of two small orthogonal matrices. Recently, Circulant matrix properties have been used in [74] to obtain a low-storage projection matrix that requires storing only d values for compressing a d dimensional vector to d bits. We present an alternative method for compressing high dimensional descriptors based on hypersphere hashing functions inspired by the Spherical Hashing method(SpH) of Heo et al [72].

Hypersphere hashing functions have been shown to outperform hyperplane

methods [11, 15, 28] for low numbers of bits by capturing the non-linearity of the data. We explore the potential of the hyperspherical hashing functions for compressing large dimensional descriptors by proposing a hierarchical approach that overcomes the computational challenges of learning spherical hashing functions for high dimensional data. We take advantage of the structural properties of VLAD and FV to split the vectors and learn spherical hashing functions for partitions. Full spherical hashing functions are constructed from the sub-spherical hashing functions in a hierarchical way using a Random-Select and Adjust(RSA) method which we also introduce in this work.

This work is organized as follows. In section 4.2, we discuss the Spherical Hashing method that defines the hypersphere hashing functions and an efficient iterative optimization scheme that achieves the hashing properties for the hashing functions. We also highlight the computational challenges of applying this technique to high dimensional vectors. In section 4.3 we introduce our Hierarchical Spherical Hashing method for compressing high dimensional descriptors. We further discuss the details about partitioning of vectors, learning sub-spherical hashing funtions and construction of full hashing functions using RSA in a divide and conquer fashion, while highlighting the computational advantages of our solution. Our experiments in section 4.4 demonstrate the performance of our approach(SpH-RSA) in comparison to existing state-of-the art methods for compressing large dimensional descriptors.

4.2 Spherical Hashing

Heo et al [72] introduce a novel hypersphere-based hashing function to map spatially coherent data points into a binary code. Their approach is motivated by the fact that a single hypersphere can enclose a closed region in d-dimensional space while it requires $d+1$ hyperplanes to define the same region, thus requiring fewer bits to encode data points. A spherical hashing function is defined by a center(c) and a threshold(t), whose value is $+1/-1$ if a point is inside/outside the hypersphere. If c_p, t_p define a hypersphere p, the corresponding hashing function h_p for a feature point $x \in \mathbb{R}^d$ that generates the p^{th} binary code is given as $sgn(t_p - dist(c_p, x))$, where $dist(.,.)$ denotes the Euclidean distance from its center to the point.

To obtain k bits, SpH learns k spherical hashing functions with the objective of satisfying the balance and independence properties of hashing. These properties play an important role in producing compact binary codes as independent hashing functions distribute points in a balanced-manner to different binary codes [75, 76]. An efficient iterative scheme was proposed to meet these hashing constraints. The algorithm starts with selection of k centers and chooses k thresholds that satisfy the balance property. With the selected thresholds and centers, the method computes the number of training points inside each hypersphere(o_i) and in each pair of hyperspheres(o_{ij}). These statistics are used to adjust the centers by applying a replusive or attractive force based on the deviation from the ideal independence criterion $n/4$ between every pair of hyperspheres, where n is the number of training samples. This process is repeated until the centers converge, but typically some

Time(d/k)	25600	12800	6400	3200
12800	-	-	-	> 8
6400	>8	>8	>8	3.5182
3200	6.2317	4.1013	3.6981	1.1891
400	0.3718	0.1034	0.0599	0.0318

Table 4.1: Training times (in hours) for learning $k = \{12800, 6400, 3200, 400\}$ spherical hashing functions for $d = \{25600, 12800, 6400, 3200\}$ dimensional vectors using [72]. We used 20000 training samples for training from the ILSVRC2010(Train) dataset. >8 means that the learning did not finish after 8 hours. '-' indicates that we could not learn the hashing functions on a single machine with 16 GB RAM, as they did not fit in memory. We see that as the number of hyperspheres to be learned and the dimension of the data increases, the training times increase exponentially.

error is allowed to avoid over-fitting. For the detailed algorithm, refer to [72].

4.2.1 Computation Challenges(d is large and $k \sim d$)

The time complexity of the above iterative process for a single iteration is $\mathcal{O}((k^2 + kd)n)$, where k is the number of bits to be learned, d is the size of the vector and n is the number of training samples. For small numbers of bits $k \ll d$, the learning algorithm is practical and very fast. To learn large number of bits ($k \sim d$ and d is large), this becomes computationally intractable.

There are two computationally intensive steps in the algorithm that inhibit its extension to higher dimensional vectors. The first is to compute distances of all feature points to k hyperspheres. The second is to compute the forces for adjusting

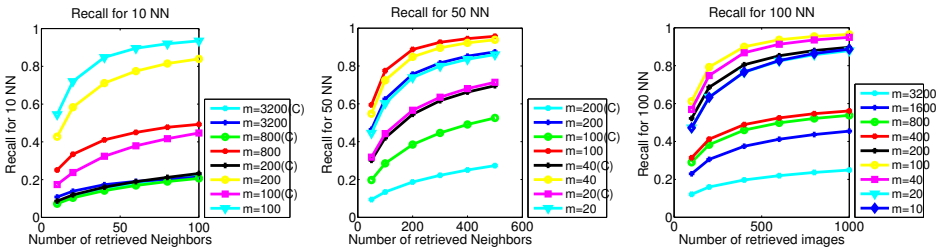

Figure 4.1: Performance of our method(SpH-RSA) and SpH-Concat on a subset of ILSVRC2010 Train dataset with 25600 VLAD vectors for varying partition size m= 3200 to 10. Plots (Left, middle) show recall for 10 and 50 ground truth neighbors while comparing our method with the concatenated bit vectors from SpH-Concat(indicated by C). Notice the poor performance of the concatenated bit vectors. Plot (Right) evaluates the results of our method for varying sub vector sizes.

the centers to satisfy the independence property. It is important to keep in mind that, the amount of training data required to learn a large number of hash functions is also very high. Such large amounts of training data with high dimensional descriptors cannot fit in memory and the algorithm to adjust the hash functions for even a single iteration becomes intractable as it suffers from the latency of reading from disk. Table 4.1 gives the training time of spherical hashing for learning varying number of hashing functions. Section 4.3.5 will discuss how we overcome these computational challenges for learning a large number of hashing functions.

4.3 Hierarchical Spherical Hashing

We alleviate the problem of learning hyperspheres for large dimensional vectors by partitioning the vectors and learning sub-hyperspheres for each partition. This

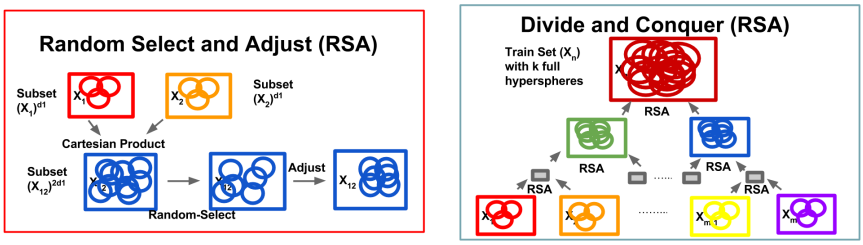

Figure 4.2: Left: RSA technique that randomly selects hyperspheres from the cartesian product of the subsets and adjusts the hyperspheres to satisfy the hashing properties. Right:RSA method applied in a Divide and conquer fashion to obtain the k full hyperspheres.

results in a feasible procedure for learning the hyperspheres in the full space. We refer to this method as Hierarchical Spherical Hashing as we start by learning sub-hypersphere hashing functions at the lower levels and extend them to higher levels using the Random Select Adjust(RSA) algorithm; we satisfy the hashing properties at each level as it is constructed in a divide and conquer style. The subsequent sections discuss the details of the algorithm.

4.3.1 Learning Sub-Hypersphere functions

Data: Let $X = \{x_1, x_2, ..., x_N\}, x_i \in \mathbb{R}^d$ denote the set of N data points of dimension d. d is large in our case. We want to compress each data point x_i to a binary vector $b_i \in \{0,1\}^k$ with $k \sim d$. Let X_n be the subset of n data points from X that we use for training.

Partitioning the data: To learn k hyperspheres for X_n, we partition each data point x_i to m subfeature vectors $\{x_{i1}, x_{i2}, ...x_{im}\}, x_{ip} \in \mathbb{R}^s$, $p = 1..m$, where

$s = \frac{d}{m}$ denotes the size of the subfeature vector. We obtain subsets(or components) $\{X_1, X_2, ..., X_m\}$, where each subset $X_p = \{x_{1p}, x_{2p}, ..., x_{np}\}$, $p = 1..m$ is the set of all p^{th} subfeature vectors of X_n. Note that we use subset or component to refer to X_p.

Learning Sub-hypersphere Hashing functions: For each subset X_i, we learn k_i hypersphere hash functions as in section 4.2. We refer to the hypersphere hashing functions learned for each subset as sub-hypersphere hashing functions. To avoid bias in learning a varying number of hash functions for each component, we learn an equal number of hashing functions for each subset, $k_i = \kappa, i = 1..m$. Therefore, training on m components produces $m\kappa$ sub-hypersphere hash functions. One could concatenate the bits obtained from the sub-hyperspheres of the m subsets to obtain a full bit-vector. We refer to this method SpH-Concat. However, this is not a good choice[1]. Figure 4.1 compares the performance of our method with SpH-Concat for different sizes of m, indicating a poor performance for the latter. We therefore employ a hierarchical approach that uses these learned sub-hyperspheres to obtain k full hyperspheres.

Size of (s, m)***:*** The parameters that influence the selection of the size of the partition(m) are the training time, memory required and performance on each subset. We choose s so as to fit the training data of a subset in memory and achieve

[1]The concatenated bit vector does not have a global relation to the full feature vector due to the de-correlation of bits between each sub-bit vector and hence is not a suitable global bit vector. Also, the independence and balance properties are only satisfied for the sub-bit vectors and not for the concatenated bit vector.

a reasonable training time and accuracy for learning sub-hypersphere hash functions using the algorithm SpH. We can at most learn up to 4000 sub-hyperspheres for a subset of dimension 4000 in typically <8 hours on a machine with 16GB RAM. Figure 4.1 shows the performance of our method on ILSVRC2010 Train dataset for various m.

4.3.2 Cartesian-product of pivots/centers

To this point we have constructed κ sub-hyperspherical hashing functions for each component. Let $\{C_1, C_2, ...C_m\}$ be the pivots(centers) learned for the m subsets, where each $C_i = \{c_{i1}, c_{i2}, ..., c_{i_\kappa}\}$, $\forall i = 1..m$, define the κ sub-hypersphere centers learned for X_i. The idea is to select k-pivots for the full space from the Cartesian product of the learned subspace pivots given as:

$$C_1 \times C_2 \times ...C_m$$

From the κ^m combinations of pivots of sub spaces, we randomly select k pivots in a divide and conquer fashion using the RSA method. It makes sense to select random k pivots from the Cartesian product based on the down-ward closure property, which is commonly used in subspace clustering algorithms [77]. This property states that *"a cluster in the sub space is a candidate for the cluster in the full feature space"*. In our case, this property suggests that any pivot choosen from the Cartesian product is a candidate pivot in the full space. We will see later that learning the full spherical hashing functions from the sub-spherical hashing functions gives a huge computational advantage as the hashing properties of the sub-spherical hashing

functions are already met.

Similarity to other methods: The idea of using the Cartesian product of sub-pivots for selection of the pivots in full space is similar to the Product Quantization method by Jegou et al [65] an Cartesian k-means [71]. While both methods use the codebook for compression, there are several differences. First, PQ obtains a full bit vector by concatenating the sub-bit vectors, whereas our full bit vector is encoded using the full spherical hashing functions learned from the sub-hyperspheres. Second, they do not use the bit vectors directly for computing distance, while ours directly computes the Hamming distance between the bit vectors, providing a computational advantage. To facilitate this, they do a lookup from pre-computed tables that contain distances between sub-codebooks, adding an additional storage cost to store these tables. Third, they select exactly one pivot from each sub-space to define the full pivot, while we select more than one pivot from each sub-space to obtain the final k pivots, increasing the representability. Finally, our pivot is always associated with a threshold, which ensures that the hypersphere hashing functions satisfy balance and independence properties, a requirement to obtain compact bit representations. There are no such guarantees for these methods [65, 71].

4.3.3 Random-Select and Adjust(RSA)

The RSA method is a two-step process that learns hashing functions of the combined subsets from the sub-hyperspheres of two components. The first is the **Random-Select Step** which randomly selects the pivots from the cartesian product

of sub-hypersphere centers and the second is the **Adjust Step** which adjusts the pivots to define the hashing functions of the combined components. Figure 4.2 illustrates the RSA algorithm.

Let $X_1 \in \mathbb{R}^s$ and $X_2 \in \mathbb{R}^s$ define two subsets of the data, from which we learned C_1 and C_2, each containing κ pivots. We learn 2κ hyperspheres for the combined set $X_{12} = \{(x_{11}, x_{21}), ..., (x_{1n}, x_{2n})\} \in \mathbb{R}^{2s}$.

1. **Random-Select Step:** This step selects 2κ random pivots from the Cartesian product of C_1 and C_2. These form the potential candidates for the pivots of the final hashing functions with the necessity of adjusting the pivots and thresholds.

2. **Adjust Step:** We use a similar method as is used for learning spherical hashing functions for selection of thresholds. The balance property is satisfied by selecting the radii that partitions the training data into two equal parts and the independence property is satisfied by applying an attractive or repulsive force in proportion to the deviation from the ideal independence criterion $\frac{n}{4}$. We do this iteratively until the hashing functions converge, thus producing a linear number of 2κ hashing functions that satisfies the hashing properties of the combined set X_{12}.

Convergence: Our goal in learning full hyperspheres from sub-hyperspheres is to obtain a computational advantage as the hashing properties are already met for the subsequent iterations. We observe that it typically takes < 2 iterations to perform RSA on the combined subsets, compared to the original SpH algorithm

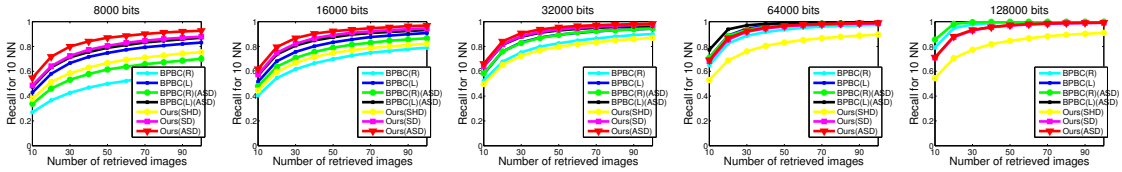

Figure 4.3: Recall plots showing the performance of Ours(Red) and BPBC(Learned and Random) methods on ILSVRC2010 Validation data set for ground truth defined at 10 Nearest Neighbors. Our method performs better for 8000, 16000 and 32000 bits for all the retrieved images in both the distance settings.

which takes at least 30 iterations(only when the training data fits in memory), thus reducing the computational time and providing a tractable solution to learn k_{12} centers.

4.3.4 Divide and Conquer

We apply the RSA algorithm in a divide and conquer style with a divide step that partitions the data and a conquer step that applies the RSA method to learn the hyperspheres for the combined partition. Figure 4.2 illustrates the process of obtaining the k full hyperspheres from m partitions using RSA in a divide and conquer fashion. We empirically observed that the adjust step in the final stage only requires the balanced property to be satisfied defining the k radii and naturally satisfies the independence property without the re-adjustment. Table 4.2 shows the number of iterations for RSA at lower levels and the total number of iterations needed to meet the convergence criteria for the ILSVRC2010 Validation data set containing 50K images.

k	#Levels	#Iter(L)	#Iter(T)	avg	std-dev
6400	4	235	242	34.47	43.84
12800	4	239	241	34.35	43.86
25600	4	238	238	33.23	43.7

Table 4.2: Table listing the number of iterations at lower level(Iter(L)) and all the levels(Iter(T)) along with the convergence criteria($avg(abs(o_{ij} - \frac{n}{4}))$ and $std - dev(o_{ij})$) for learning $k = \{6400, 12800, 25600\}$ hashing functions from 10K train samples of ILSVRC2010 Validation dataset(50K images, $m = 10$). Allowing 10% error on avg and 15% tolerance on $std - dev$, our method has met the convergence criteria for all the bits in few iterations. We observe that only an additional $7, 2, 0$ iterations are needed to meet the convergence at higher levels (given, ~ 20 iterations for each component in the lower level).

4.3.5 Computation time

The computational complexity of our algorithm is $O((\kappa^4 m + log(k/s)kd)n)t$, where t is the total sum of the iterations at all levels. The complexity is polynomial in terms of the number of hyperspheres learned in the lowest level as we typically observe that t is very low. Using our method, we reduce the computational burden of computing the force matrix for large numbers of bits. However, we still need to compute distances to the k-matrix at each iteration. This is only computed in subsets for learning m submodels and is relatively fast.

We notice from Table 4.2 that no additional iterations are required to satisfy the hashing properties for obtaining full hyperspheres from sub hyperspheres for

learning 25600 bits from 25600 VLAD vectors. This emphasizes that our approach succeeded in obtaining the hyperspheres that naturally satisfy the independence properties in the full space by learning only sub-hyperspheres in sub spaces. We further observe that excluding the independence step at each conquer step would only require us to compute a single balancing step at the root level for learning k pivots in the full space, decreasing the computation and storage cost drastically. The computational complexity of computing only one single balance step for the full space is $O((\kappa^2 m + \kappa d)nt_{low} + kdn)$, where t_{low} is the average of the iterations for computing sub-hypersphere hashing functions at the lowest level. Therefore, we obtain a feasible solution to train the SpH-RSA model with complexity $O(\kappa^2 mn)t_{low}$. The storage cost for such a model with k full hyperspheres is $O(d\kappa + k)$, where the first term $d\kappa$ refers to storing the $m\kappa$ sub-hyperspheres and the second term k corresponds to the k thresholds of the hashing functions. We use this model in our experiments.

4.4 Experiments

We evaluate our method on the following datasets: ILSVRC2010 Validation and Train datasets which contain 50,000 and 1.2 Million images and Holidays dataset with 1M Flickr distractors containing 1491 and 1 Million images respectively. We download these publicly available datasets [44] and [78] which already contain the computed dense sift descriptors. We use VLAD descriptors [65] for applying our compression techniques. We power normalize ($\alpha = 0.5$) the VLAD vectors and

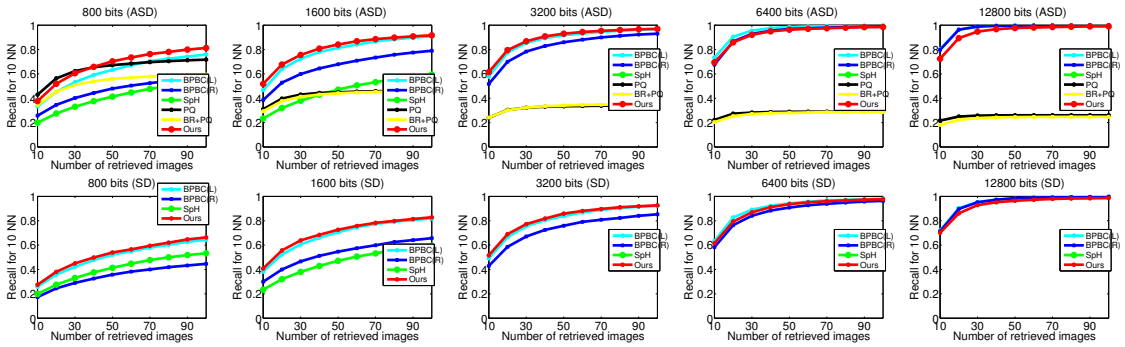

Figure 4.4: Recall plots showing the performance of Ours(Red) and several state-of-the art methods on Holidays+Flickr 1M data set for ground truth defined at 10 NNs. Top Row shows the performance of the methods using Assymetric Distance and the Bottom Row using Symmetric Distance. We see that our method consistently performs better for 1600 and 3200 bits at all the retrieved images in both the distance settings

L2-normalize the vectors to obtain the best performance for retrieval [68]. We do not need to mean-center the vectors as the VLAD vectors naturally have a mean of zero.

4.4.1 Datasets, Evaluation Protocol

We apply our method on the ILSVRC2010 Validation and Train datasets where each image is represented by a VLAD vector of size 128000 and 25600 with a codebook of size 1000 and 200 respectively. For both datasets, we randomly select 500K SIFT descriptors to learn a codebook. We partition the data in two parts:a query partition containing 500 queries and a non-query partition containing the remaining images. We use the non-query partition for retrieval and training purposes. From the non-query partition, we select 10K and 20K training samples to learn the full spherical hashing functions for the Validation and Train datasets respectively.

We also evaluate our method on the Holidays dataset [78] containing 1491 images with 500 predefined queries. We combine the rest of the 991 images with 1M Flickr distractor images for retrieval and training(20K). We use the visual dictionaries learned from independent Flickr60K images for these experiments. Specifically, we compute the VLAD vector of size 12800 (128x100) using a codebook of size 100.

We evaluate the performance of our method and the following state-of-the art methods by calculating recall for upto 100 retrieved images for top-10 Nearest Neighbors. For each query, we obtain the ground truth based on Euclidean distance of their VLAD vectors.

4.4.2 Comparison methods

We compare our methods to the following state-of-the art approaches: BPBC(Random), BPBC(Learned) [73], Product Quantization(PQ) [70] and Spherical Hashing(SpH) [72] for full vectors. CBE [74], a recent state-of-the art approach also compresses high dimensional vectors to large number of bits with a low storage cost for projection matrix. However, it mimics the performance of BPBC for compressing to both high and low number of bits and our intention is to boost the performance for low number of bits using compact hyperspherical hashing functions. So, we do not include this method in our comparison. In our experiments, we compress the VLAD vector to $k_{ILSVRC2010_{Val}} = \{128000, 64000, 32000, 16000, 8000\}$ bits for ILSVRC2010 Validation, $k_{ILSVRC10_{Train}} = \{25600, 12800, 6400, 3200, 1600\}$ bits for ILSVRC2010 Train and $k_{Hol+Flickr1M}= \{12800, 6400, 3200, 1600, 800\}$ bits for Holidays+Flickr1M datasets.

1. **BPBC:** For obtaining a compression of k bits, both the learned ($BPBC_L$) and random versions ($BPBC_R$) learn two bilinear projection matrices $R_1^{c\times k_1}$ and $R_2^{128\times k_2}$, where $k_2 = \frac{k}{k_1}$ and c is the codebook size. To compress a d dimensional vector to $\{d, \frac{d}{2}, \frac{d}{4}, \frac{d}{8}, \frac{d}{16}\}$ bits, we chose $k_1 = \{c, \frac{4c}{5}, \frac{c}{2}, \frac{2c}{5}, \frac{c}{4}\}$ to ensure the method's best performance as suggested in [73].

2. **PQ:** We obtain binary codes by applying Product Quantization(PQ) to both the original and rotated vectors(BR+PQ). We chose to rotate the vectors as per [70] to balance the bias of the vectors. Since, rotating the full vectors

is an expensive operation in terms of both storage and computing rotated vectors, we bilinearly rotate the vectors as suggested in [73]. We then apply PQ methods by chosing the subvector size as $s = 8$ and learning m_c clusters for each sub vector as in [65]. For all the datasets, we learn codebook of size $m_c = \{256, 128, 64, 32, 16\}$ from each subset. For PQ and BR+PQ, we use only Assysmetric Distance (distance between the query and codebook vectors) (ASD) to obtain the neighbors, as this is shown to always outperform the Symmetric Distance(SD).

3. **SpH:** We also learn directly full spherical hashing functions on the full VLAD vectors. Due to the limitation of training for large number of hash functions, we only learn the spherical hashing functions for lower numbers of bits $k_{Hol+Flickr1M_l} = \{1600, 800\}$ for the Holidays+Flickr1M data set only. To be consistent, we use the same training set we used to learn the sub-hypersphere hashing functions.

4.4.3 Results

Parameters for SpH-RSA: The following properties are common for all the datasets. We chose $s = 256$ and $\kappa = 1280$ as suggested from our experiments in Figure 4.1.To select the hypersphere hashing functions for lower number of bits, we chose the hashing functions that most satisfy the hashing properties. The results reported are an average of 5 iterations to be consistent with our random selection scheme. Typically, we observed a small standard deviation value of 0.0253, 0.048

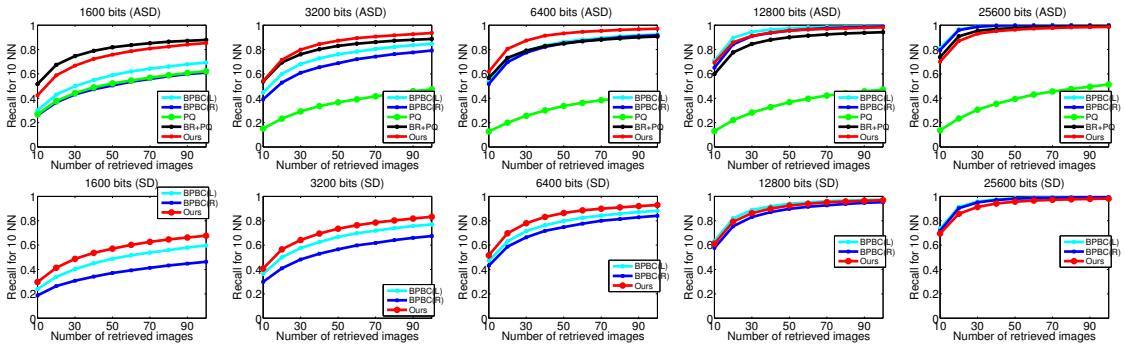

Figure 4.5: Recall plots showing the performance of Ours(Red), BPBC and PQ based methods on ILSVRC2010 Train data set for ground truth defined at 10 NNs. Top Row shows the performance of the methods using Assymetric Distance and the Bottom Row using Symmetric Distance. We see that our method consistently performs better for 3200 and 6400 bits at all the retrieved images in both the distance settings.

and 0.0361 for ILSVRC2010 Validation, Train and Holiday datasets for learning k bits.

Symmetric/Assymetric Distance: As PQ based methods(PQ and BR+PQ) perform better with using Asymmetric Distance, we report the results on both Symmetric and Assymetric Distance. Symmetric distance(SD) is the traditional Hamming distance between the query bit code and the retrieval image bit code. This can be efficiently obtained by an XOR Operation and POP COUNT. Assymetric distance(ASD) is obtained by computing the distance between the projected query vector and the bit code. For SpH and our method, the projected vector of a query q: $q_p = [q_{p1}\ q_{p2}\ ....\ q_{pk}]$ is obtained as $q_{pl} = t_l - dist(c_l, q)$, where $dist$ is the euclidean distance of the hypersphere(l) with center(c_l) and threshold t_l to the query q, $l = 1..k$. For BPBC, the projected vector is obtained as ${R_1}^T q R_2$. ASD is now given as $dist(q_p, b) = ||q_p|| + ||b|| - 2q_p b^T$, which requires only computation of the dot product of the projected query vector(q_p) and the binary code(b), as the norm of the query and binary codes are constant.

ILSVRC2010 Validation: Figure 4.3 shows the performance of our method for compressing 128000-dim VLAD vectors to $k_{ILSVRC2010_{Val}}$ bits. We observe that our method (SpH-RSA) using both Symmetric and Assymetric distance outperforms the BPBC(Random and Learned Versions) for 8000, 16000 and 32000 bits and performs very similar to the learned method of BPBC for 64000 and 128000 bits at recall for top 100 retrieved images. Our results emphasize that the compact hyperspherical hashing functions learned significantly improve the performance at lower numbers of bits than the traditional hyperplane hashing functions($\sim$ 10%

for 8000 bits). For this dataset, we also calculated the Spherical Hamming Distance(SHD) [72], obtained as the ratio of Hamming distance and common number of +1 bits. Contrary to the results in Spherical Hashing method [72], SHD does not provide compact distance in our case degrades the performance. This may be because all the L2-normalized VLAD vectors are lying on the surface of a unit sphere of radius 1 and the ratio of the average max distance of the nearest neighbors to distant neighbors is close to 1.

Holidays+Flickr 1M: Figure 4.4 shows the performance of our method for compressing 12800-dim VLAD vectors to $k_{Hol+Flickr1M}$ bits. Our method(SpH-RSA) performs very similar to the BPBC(Learned) method for all bits, but outperforms PQ and BR+PQ except for 800 bits. Surprisingly, as the size of the codebook increases the performance of PQ and BR+PQ decreases. This might be because of learning too many clusters and forming different clusters with similar properties, causing an over-segmentation problem. Gong et al [73] reported a similar trend for PQ and BR+PQ in their results. We achieve a 2% and 5% improvement over BPBC(L) for SD and ASD when compressing to 800 bits(both recall@10 and recall @100). We have also compared our method with the Spherical Hashing method (SpH) learned from full VLAD vectors, and we observed a low recall for the $k_{Hol+Flickr1M_l}$ bits using SpH.

ILSVRC2010 Train: Figure 4.5 shows the performance of our method for compressing 25600-dim VLAD vectors to $k_{ILSVRC2010_{Train}}$ bits. We observe that our method (SpH-RSA) significantly outperforms the BPBC(Learned and Random) methods for 1600, 3200 and 6400 bits for both the Symmetric and Assymetric

Distance based results. Similar performance is obtained for recall@100 with the BPBC(Learned) method for 12800 and 25600 bits and ours consistently improves over the random Bilinear Projections method in the two distance settings. Comparing the performance to the PQ based methods (BR+PQ and PQ), our encoding schemes obtain higher recall for all the bits except 1600 bits. Notice that the performance of PQ methods does not increase with an increase in the number of codebooks learned as also seen in [73]. This emphasizes that the parameters of s and m for PQ need to be carefully selected to obtain optimal performance, while our method is robust to these choices. Finally, we report approximately 4% and 14% improvement over the learned BPBC scheme using the Symmetric and Assymetric Distance respectively(recall@100 for 1600 bits).

Training times: By partitioning the problem, we have proposed a practical solution to learn hypersphere hashing functions for high dimensional descriptors. Our approach takes approximately 60 and 40 minutes for training 128000 and 25600 hyperspheres on the training sets of ILSVRC2010 Validation and Train datasets. Training times are calculated for learning $m\kappa$ sub-hyperspheres, m=$\{200, 100\}$ on a single machine with 16GB RAM, Intel Xeon Quad CPU @ 2.66 GHz.

4.5 Conclusion

We have proposed an efficient method for learning hypersphere hashing functions for high dimensional data. Our method employs a hierarchical procedure that

This work is supported by NSF EAGER grant: IIS1359900, Scalable Video Retrieval.

learns hash functions for paritions of the data and utilizes the RSA method to obtain the hashing functions for the full space. We have shown a significant improvement of 10% and 14% over the BPBC(learned) method for a compression ratio of 512 for ILSVRC2010 Validation, Train datasets. We also report increases in performance over the state of the art methods for compression ratios of 256 and 128 and demonstrated similar performance for compression ratios of 64 and 32.

Chapter 5: **Zero-shot Event Detection using Multi-modal Fusion of Weakly Supervised Concepts**

5.1 Overview

Popular websites such as YouTube, Google images, and Flickr contain large volumes of image and video data from a multitude of consumer devices such as digital and cellphone cameras. Technologies that can rapidly analyze such content and detect salient concepts and events have several compelling applications. Significant progress has been made in developing such technologies and the core of most state-of-the-art methods is based on the *bag-of-words* model [79]. Here, we first extract low-level features that capture salient gradient [80, 81], color [82], or motion [83, 84] patterns, project them to a pre-trained codebook in the same feature space, and then aggregate the projections to get the final image or video level feature vector. Classifiers, typically kernel support vector machines (SVM), are then trained using labeled data. This approach requires a large number of training examples for each class of interest and performance decreases rapidly as the training set size decreases.

In this work, we study the problem of video classification using only a textual description of the events of interest, without exemplar videos pertaining to

the events. This zero-shot framework, where we perform video classification with zero training samples, goes beyond traditional zero-shot problems such as described in [85], where an existing set of classes with training data is adapted to an unseen class. We pose this difficult problem of video classification as a retrieval task, where an event is described as a query defined by a set of concepts, e.g. the event "driving a car" described by the set of concepts "drive, car, road, person, face." We aim to retrieve videos that are most similar to the query, where the similarity score is treated as the confidence of the video belonging to that event.

Our approach to zero-shot learning is to first transform both video and query text to a high-dimensional concept space before computing similarity in that space. For the query, we apply text processing techniques to obtain a vector of salient words and phrases describing the event. For the video, we apply a bank of concept detectors to obtain a textual representation of the video using a vector of detection scores. Since we have no prior knowledge of the events of interest, we need a very large set of generic concept detectors in order to provide semantic coverage of all possible queries. To address this challenge, we utilize multiple concept detectors from different modalities: visual features, including video concepts and multiple query fusion [86] of multiple features described in this work, in addition to off-the-shelf detectors such as Classemes [4], ObjectBank [5] and SUN attributes [43]; audio information from concepts learned on low-level Mel-frequency Cepstrum Coefficients(MFCC) features; and text from video text and speech transcriptions.

Once we represent both query and videos as vectors of concept scores, we can compute similarities to retrieve relevant videos. A key challenge here is the mismatch

between query and video concept vocabularies. We utilize a text expansion based method to project query and video concept vectors to a common high-dimensional concept space where they are compared, using the large text corpus Gigaword [87] to learn this projection matrix. Finally, we fuse retrievals from each of the features and modalities using a simple linear combination to exploit the complementary nature of the different modalities and concept vocabularies.

The work is organized as follows: in Section 5.2, we discuss related approaches to similar problems. In Section 5.3, we present an overview of our zero-shot learning framework. Section 5.4 describes the features we extract from video and Section 5.5 outlines the combination of these features. We report experimental results in Section 5.6, and discuss our conclusions in Section 5.7.

5.2 Related Work

Extensive research has been performed in recent years on effective representation and classification of images and videos. The first step in most techniques is to extract *low-level features* from local spatial or spatio-temporal patches. Popular features include grayscale appearance features such as SIFT [80] and SURF [81], color features such as Color SIFT [82], and motion features such as STIP [83] and dense trajectories [84]. These typically extract thousands to millions of feature vectors per image or video. They are aggregated to a single fixed dimensional representation by a sequence of *coding* and *pooling* steps. Possible coding techniques include *Hard Quantization* [79], *Soft Quantization* [88], *Sparse Coding* [89] and *Fisher Vec-*

tors [90], using a codebook trained in an unsupervised manner from a large set of feature vectors. The coded features are then aggregated, typically using average or max pooling, and classified typically using support vector machines (SVM).

While this approach has shown strong results given a large training set, performance degrades rapidly as the amount of training data decreases and the method does not generalize to previously unseen events. Only limited attention has been paid to this challenging problem and most existing approaches introduce an intermediate layer of semantic concepts, which are then used to describe novel classes. Semantic output codes (SOC) are proposed in [85] to extrapolate novel classes by utilizing a knowledge base of semantic properties of known classes. A large scale ontology is used in [91] to learn visual relationships between objects, while [92] uses knowledge transfer between object classes. An online incremental attribute based zero-shot learning approach is presented in [93], while a max-margin formulation is proposed in [94] for zero-shot multi-label classification where the label correlations on the training set differ significantly from the test set. A constrained optimization formulation that combines regression and knowledge transfer based functions has recently been proposed in [95].

All of these techniques rely on extrapolating from an existing set of classes and training data. The more difficult task of performing video retrieval and classification with no prior event knowledge or training data has been addressed only recently. In contrast to [96], we introduce several ways to generate a large visual and audio concept lexicon without prior knowledge of the event classes, and present a simple unified framework for effectively combining visual, audio, and textual information.

While we are not able to benchmark our method against [96] since we do not have access to their concept lexicon or data partitions, our results in the TRECVID evaluation (Section 5.6.6) compare favorably to systems using similar approaches.

Video retrieval using semantic similarity has previously been explored in [97, 98]. However, these approaches focus on highly structured broadcast data, where a small 374 concept pool [97] can be adequate. In contrast, we focus on more challenging unconstrained web data where leveraging multiple modalities and larger concept banks is important to build a robust system. While [97, 98] both use a pre-defined concept ontology, we demonstrate the benefit of training in-domain detectors in a data driven manner by discovering concepts from free form text descriptions.

There has also been an increasing interest in joint modeling of text and visual features [99], which can then potentially be used to generate a text description of query images [100–103] and videos [104, 105]. A large scale study of the relationship between semantic similarity of classes and confusion between them is presented in [106]. In [107], a large text corpus is used to learn a semantic space using word distributions and a separate model is trained for seen and unseen classes. However, given the training data limitations in our problem, we constrain our focus to attribute mappings produced using off-the-shelf features [4, 5, 43], novel concept banks developed with video-caption pairs similar to [103], and speech and video text output.

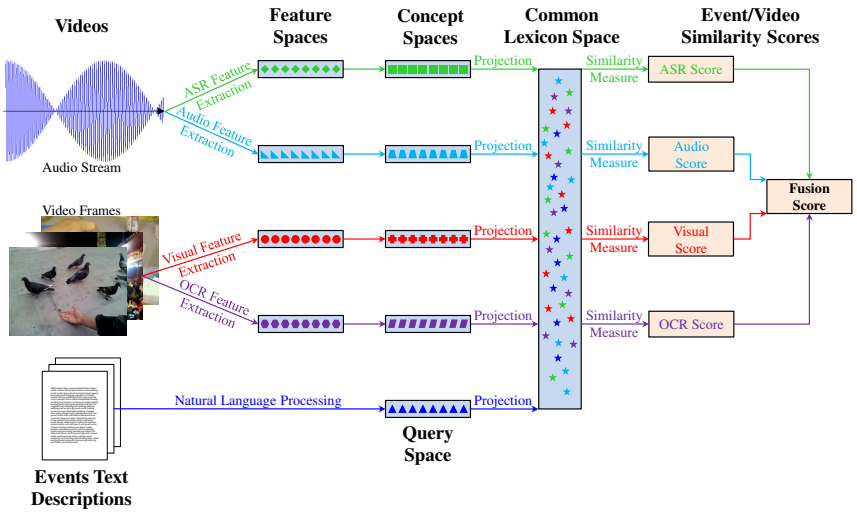

Figure 5.1: Overview of the proposed multi-modal zero-shot learning approach.

5.3 Zero-shot Learning Framework

Figure 5.1 displays an overview of our multi-modal zero-shot learning approach, which involves applying C different concept banks on each video v. Let $L = \{c_{l_1}, \ldots, c_{l_K}\}$ be a lexicon defined by K concepts c_{l_k} for concept bank $l \in [1, \ldots, C]$. Each concept bank provides a K-dimensional vector of detection scores $\vec{d_v} = [d_{l_1} \ldots d_{l_K}]^T$ for each video $v = 1 \ldots V$, that is ℓ_2-normalized; i.e., $\|\vec{d_v}\|_2 = 1$. Given a query $Q = \{c_{q_1}, \ldots c_{q_N}\}$ defined by N concepts c_{q_n}, we aim to retrieve videos that are similar to the query.

5.3.1 Basic Similarity Computation

We first present a direct model to measure video-query similarity. In this model, we compute the similarity score $S_Q(v)$ between a query Q and a video v as

a sum of the concept scores of the lexicon that match the query concepts:

$$S_Q(v) = \frac{1}{K} \sum_{k=1}^{K} d_{l_k} \mathbb{1}_Q(c_{l_k}) \tag{5.1}$$

where $\mathbb{1}_Q(c_{l_k})$ is an indicator function of the presence of concept c_{l_k} in query Q.

This baseline system is very precise for efficient concept detectors. We expect the system to perform well when there is a large match between the query and video concepts, but lexicon coverage of the query will limit recall while noise in the video concept detections will degrade precision.

5.3.2 Expansion-based Similarity Computation

To address the issue of vocabulary mismatch between query and video, we use an alternative model to measure video-query similarity. In this model, concepts are expanded and projected to a common global concept space defined by the lexicon L. The goal is to propagate existing confidence scores to semantically similar concepts using the knowledge from a text corpus (like Gigaword) to estimate similarity. Let $G : (c_1, c_2) \longrightarrow s$ be a text model that measures the similarity $s \in [0, 1]$ between two concepts c_1 and c_2. Let an item I in the database be represented by a set of triplets describing the concept name, its confidence score, and its index in the lexicon L. The expansion-based projection method is given in Algorithm 2.

This algorithm obtains the projected vector $\vec{f}$ of an item I in two steps for each concept. The first step finds the top T similar concepts using the model G. The second step boosts the scores of the similar concepts for an item by the amount of similarity between the concepts. The final feature vector $\vec{f}$ is then normalized for

Algorithm 2 Expansion-based projection.

Given an item $I = \{(c_1, s_1, i_1), \ldots, (c_N, s_N, i_N)\}$.

Let $\vec{f} \in \mathbb{R}^K$ be the projected feature vector of item I for L.

Initialization: $f_k = 0$ for $k = 1 \ldots K$.

for each (c, s, i) in I **do**

$f_i \leftarrow f_i + s$

Find the top T similar concepts of c in G, given as

$$I_T = \{(c_1, s_1, i_1), \ldots, (c_T, s_T, i_T)\},$$

where $s_t = G(c, c_t)$.

Update the feature for the similar concepts:

for each (c_t, s_t, i_t) in I_T **do**

$f_{i_t} \leftarrow f_{i_t} + s \cdot s_t$

end for

end for

Normalize the feature vector $\|\vec{f}\|_2 = 1$.

comparison purposes.

Algorithm 2 is applied to expand both the query and database concepts to a common lexicon space. Query concept confidences are given as 1, while database concept confidences are given by the output of the concept detectors. Once the expanded feature vectors $\vec{f_Q} \in \mathbb{R}^K$ representing the query Q and $\vec{f_v} \in \mathbb{R}^K$ representing the video v have been obtained, the similarity between the query Q and the video v is computed as

$$S_Q(v) = \vec{f_Q}^T \vec{f_v}. \tag{5.2}$$

Note that other similarity measures may also be considered (e.g., Laplacian or RBF kernels), although in our experiments we find that (5.2) has the best performance.

5.4 Video Feature Extraction

Since existing concept banks are generally trained on out of domain data and may not contain a large enough vocabulary to cover possible queries, we propose multiple methods to rapidly learn new concept detectors with easily collected data from readily available in-domain and web sources.

5.4.1 Weakly Supervised Concepts (WSC)

We train a set of WSCs for concept detection in videos using the following steps:

5.4.1.1 Data Collection and Concept Discovery

We collect a set of videos with free-form text descriptions of their content. Such data is widely available online in websites such as YouTube and also in the *research* set of the considered TRECVID MED dataset. We apply standard natural language processing (NLP) techniques to clean up the annotations, including removal of common stop words and stemming to normalize word inflections. The remaining vocabulary is taken as our concept dictionary.

5.4.1.2 Low-level feature extraction

For each video in the collected corpus, we extract the following set of low-level visual and audio features:

D-SIFT [89]: This is a dense version of SIFT where, instead of detecting interest points, the 128-dimensional feature vectors are extracted at uniformly-sampled locations covering the whole image. D-SIFT typically generates 3× the number of points produced by SIFT [80] and has been shown to outperform SIFT for image classification [89].

Dense Trajectories (DT) [84]: This feature represents the video using dense optical flow trajectories. Histogram of oriented gradients (HoG) and motion boundary histograms (MBH) are extracted from the local spatio-temporal neighborhood of each track to capture salient appearance and motion patterns respectively.

MFCC [108]: These popular audio features are extracted from overlapping 29 ms frames at a rate of 100 frames per second. From each frame, we compute 14 mel-frequency warped cepstral coefficients. The resulting 45-dimensional feature vector captures the short-time spectral structure of the audio stream.

For each of the above low-level features, we first apply principal component analysis (PCA) to reduce the dimensionality and whiten the feature vectors. For each video, we then obtain a set $X = \left\{\mathbf{x}_t \in \mathbb{R}^D,\, t = 1 \ldots T\right\}$ of T low-level low-dimensionality feature descriptors. We assume that these features are distributed

according to a Gaussian mixture model (GMM) with diagonal covariance matrix:

$$p(\vec{x}_t|\Lambda) = \sum_{k=1}^{K} w_k \mathcal{N}(\vec{x}_t; \boldsymbol{\mu}_k, \boldsymbol{\sigma}_k^2), \text{ for } t = 1 \dots T. \tag{5.3}$$

The GMM parameters

$$\Lambda = \left\{ w_k \in [0,1], \boldsymbol{\mu}_k \in \mathbb{R}^D, \boldsymbol{\sigma}_k \in \mathbb{R}^D, \, k = 1 \dots K \right\}$$

are learned on a training set through maximum likelihood estimation. We then consider the Fisher vector encoding as proposed in [109] and represent each video by the normalized gradients of the GMM log-likelihood $\mathcal{G}_X^{\boldsymbol{\mu}_k} \in \mathbb{R}^D$ and $\mathcal{G}_X^{\boldsymbol{\sigma}_k} \in \mathbb{R}^D$ with respect to the Gaussian mean $\boldsymbol{\mu}_k$ and standard deviation parameters $\boldsymbol{\sigma}_k$, respectively. For $k = 1 \dots K$, these D-dimensional normalized gradients are defined as

$$\mathcal{G}_X^{\boldsymbol{\mu}_k} = \frac{1}{T\sqrt{w_k}} \sum_{t=1}^{T} \gamma_k(\vec{x}_t|\Lambda) \left(\frac{\vec{x}_t - \boldsymbol{\mu}_k}{\boldsymbol{\sigma_k}} \right) \tag{5.4}$$

$$\mathcal{G}_X^{\boldsymbol{\sigma}_k} = \frac{1}{T\sqrt{2w_k}} \sum_{t=1}^{T} \gamma_k(\vec{x}_t|\Lambda) \left[\frac{(\vec{x}_t - \boldsymbol{\mu}_k)^2}{\boldsymbol{\sigma}_{\boldsymbol{k}}^2} - 1 \right], \tag{5.5}$$

where the posterior probability

$$\gamma_k(\vec{x}_t|\Lambda) = \frac{w_k \mathcal{N}(\vec{x}_t; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{l=1}^{K} w_l \mathcal{N}(\vec{x}_t; \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}$$

is the soft assignment of the feature descriptor $\vec{x}_t$ to the k-th Gaussian cluster. The final Fisher vector is the concatenation of the K D-dimensional normalized gradients $\mathcal{G}_X^{\boldsymbol{\mu}_k}$ and $\mathcal{G}_X^{\boldsymbol{\sigma}_k}$, and is thus of dimension $2KD$.

Vector multiplications and divisions are element-wise operations here.

5.4.1.3 Classifier Training

For each concept identified in Section 5.4.1.1, we collect all videos for which that concept occurs in the text caption, and utilize them as our positive training set, with the remaining videos considered as negatives. We then train RBF kernel-based support vector machine (SVM) classifiers using the Fisher vectors representing the videos. We train a set of concept detectors for each of the low-level features (D-SIFT, DT, MFCC) described in Section 5.4.1.2.

5.4.1.4 Weakly Supervised Concept Feature

Given a video, we produce a compact representation by concatenating the detection scores of our concept detectors. We use this feature vector for event detection and refer to this representation as *WSC*, for weakly-supervised concepts.

5.4.2 Concept Training using Web Data

In addition to the concept detectors trained using the research set described in Section 5.4.1.1, we also train detectors using data downloaded from the web. For each concept identified in Section 5.4.1.1, we downloaded the top 100 retrievals from Google images and thumbnails for the top 50 retrievals from YouTube. We then train WSCs with this data using the same approach as described in Section 5.4.1. We call the WSCs trained using the TRECVID research set, Google images and YouTube thumbnails as $\text{WSC}_{\text{TRECVID}}$, $\text{WSC}_{\text{Google}}$ and $\text{WSC}_{\text{YouTube}}$ respectively.

5.4.3 Concept Distance Features

We also introduce a novel concept distance (CD) based feature. Let C denote the set of concepts identified from the text annotations in Section 5.4.1.1. For each concept $c \in C$, let V_c denote the set of videos in the research set containing the concept. Let $\vec{x}_i$ denote the low-level feature based vector extracted for video i. Then, we compute the feature vector $\vec{y}_c$ for the concept c as:

$$\vec{y}_c = \frac{1}{|V_c|} \sum_{i \in V_c} \vec{x}_i. \tag{5.6}$$

Given a new video v and its low-level feature vector $\vec{x}_v$, we obtain the CD feature vector by computing the distance to each $\vec{y}_c$ in (5.6) and concatenating:

$$\vec{CD}_v = \left[\|\vec{x}_v - \vec{y}_1\|_2 \ \ldots \ \|\vec{x}_v - \vec{y}_{|C|}\|_2 \right]^T. \tag{5.7}$$

In our experiments, we use D-SIFT, DT and MFCC low-level feature vectors. The proposed feature vector builds on multiple-queries (MQ) [86] and the query expansion [110] based techniques proposed previously. While these approaches identify relevant videos at query time and use the retrievals to expand the concept set or training set, we use a static set of concept vectors $\vec{y}_c$ and compute distances at query time to these vectors.

5.4.4 Off-the-shelf Concept Detectors

We also test three off-the-shelf concept detectors that have been used in recent literature:

Classemes [4]: This is a bank of concept detectors trained on images. These were

chosen using a large ontology of visual concepts. Given an image or a video frame, the application of all these detectors yields a 2,659-dimensional vector of detection scores.

ObjectBank [5]: Here, we use a spatial pyramid representation of images and produce detection confidence scores at different scales and spatial pyramids for each concept. The concept detectors are trained using linear SVMs and an image is represented by concatenating the detection scores of different concepts at different scales and spatial pyramids.

SUN Attributes [43]: The SUN attribute set contains detectors for 102 scene attributes that were specified using crowd sourced human studies.

We apply each of these concept detectors on a set of frames uniformly sampled from a video and then average the detection scores across the video to get the final video-level feature vector.

5.4.5 Automatic Speech Recognition (ASR)

We use GMM-based speech activity detection (SAD) and a hidden Markov model (HMM) based multi-pass large vocabulary ASR to obtain speech content in the video, and encode the hypotheses in the form of word lattices.

We first extract MFCC features from the audio stream. Then, the speech segments are identified by using a speech activity detection (SAD) system that employs two GMMs, for speech and non-speech observations respectively. The SAD model incorporates video clips with music content to enrich the non-speech model, in order

to handle the heterogeneous audio in consumer video. Given the automatically detected speech segments, we then apply a large-vocabulary ASR system to the speech data to produce a transcript of the spoken content. The system is adapted from an ASR system trained on English Broadcast News, and updated with MED 2011 descriptor files [111], relative web text data, and the small set of annotated consumer video data. We evaluated the ASR model on a held-out set of 100 video clips and achieved a Word Error Rate (WER) of 35.8%. The system outputs not only the 1-best transcripts but also word lattices with acoustic and language model scores.

After basic processing to remove stop words and normalize word inflections, the word lattice posteriors are used to generate the concept score vectors used in the zero shot projection system.

5.4.6 Optical Character Recognition (OCR)

Our OCR system recognizes text in bounding boxes from a video text detector using an HMM-based multi-pass large vocabulary OCR system. Similar to our ASR system, word lattices are used to encode alternative hypotheses. We leverage a statistically trained video text detector based on SVM to estimate video text bounding boxes.

Text candidate regions are first selected using Maximally Stable Extremal Regions (MSER) and filtered using an SVM with rich shape descriptors such as Histogram of Oriented Gradients (HoG), Gabor filter, corners and geometrical features. Candidate regions are then grouped to form word boundaries, and detected words

are binarized and filtered before being passed to the HMM-based OCR system for recognition. The OCR system finds a sequence of characters that maximizes the posterior, by using glyph models (similar to the acoustic models in ASR), a dictionary and N-gram language models. The word precision and recall of our system measured on a small consumer video dataset is 14.7% and 37% respectively.

Since the video text content presents itself in various forms, such as subtitles, markup titles and in-scene text, it is much more challenging than conventional scanned document OCR. To address these challenges, we consider two versions of OCR: one which utilizes the dictionary and N-gram language model, and one which is character-based. While the language model corrects character-level transcription errors, it also introduce errors when falsely correcting out of vocabulary words. For the word model OCR output, we generate a concept score vector from the word lattice posteriors in the same way as ASR. For the character based model, we estimate word posteriors by smoothing character errors across adjacent video frames to produce a concept score vector. In our experiments we find the character model to be slightly better for video than the word model, as detailed in Section 5.6.

5.5 Fusion

State of the art systems for standard event detection with training data have shown fusion of multiple features and modalities to be crucial for improving performance [112]. Fusion is especially important for the zero-shot problem, due to the sparse occurrence of speech and video text content, as well as the limited vocabulary

intersection between a given concept bank and query. While we do not have any training data on which to learn parameters for more sophisticated fusion methods, we find that simple score averaging works well to exploit the complementary information in various systems. We further see some benefit to manually increasing the weights of the higher precision ASR and OCR systems in fusion, and use a linearly weighted score combination for all fusion experiments below.

5.6 Experiments

We test our approach on the large collection of consumer web videos from the TRECVID MED 13 [3] dataset. The task is to retrieve videos containing one of 20 diverse high-level multimedia events, each described by a short text document of ∼250 words. The dataset provides a *research* set that contains ∼12,000 background videos and no exemplars of the events of interest. We use this research set to learn our $\text{WSC}_{\text{TRECVID}}$ and CD features. We report on the designated *MEDTest* set containing ∼25,000 videos. More details of the events and data partitions may be found in [3].

5.6.1 Comparison of Similarity Computation

Table 5.1 compares the two methods of query-video similarity computation discussed in Section 5.3.1 and Section 5.3.2 for the best feature in each modality. We observe that expansion consistently improves over the simple approach. We observed similar gains from using projection based features in fusion, and thus we

Feature	Basic (MAP)	Expanded (MAP)
ASR	3.27%	**3.66**%
OCR (character)	4.43%	**4.72**%
CD^{MFCC}	**1.04**%	**1.04**%
$\text{WSC}^{\text{D-SIFT}}_{\text{YouTube}}$	3.42%	**3.48**%

Table 5.1: Mean average precision (MAP) comparison between basic (5.1) and expanded (5.2) query-video similarity computation for our single best ASR, OCR, audio, and visual features.

use the expansion-based approach in all experiments below.

5.6.2 Comparison of Visual Features

In these experiments, we compare our proposed WSC and CD features to several off-the-shelf detectors. Table 5.2 summarizes our results. Here, $\text{WSC}^{\text{D-SIFT}}_{\text{YouTube}}$ refers to the weakly supervised concept features trained using D-SIFT features extracted on pre-downloaded YouTube thumbnails. Overall, the $\text{WSC}^{\text{D-SIFT}}_{\text{YouTube}}$ feature has the strongest performance, while the off-the-shelf detectors are significantly weaker than our proposed approaches. A possible reason for this is the large domain mismatch between the data used for training them and the video data. The same issue could explain the weaker performance of the $\text{WSC}_{\text{Google}}$ features compared to $\text{WSC}_{\text{TRECVID}}$ and $\text{WSC}_{\text{YouTube}}$ due to the domain mismatch between images

Feature	MAP	AUC
SUN [43]	0.48%	0.605
ObjectBank [5]	0.77%	0.592
Classemes [4]	0.84%	0.630
$CD^{\text{D-SIFT}}$	1.71%	0.770
CD^{DT}	2.28%	**0.779**
$WSC^{\text{D-SIFT}}_{\text{TRECVID}}$	1.92%	0.735
$WSC^{\text{DT}}_{\text{TRECVID}}$	2.76%	0.726
$WSC^{\text{D-SIFT}}_{\text{Google}}$	1.21%	0.543
$WSC^{\text{D-SIFT}}_{\text{YouTube}}$	**3.48**%	0.729

Table 5.2: Comparison of mean average precision (MAP) and area under the curve (AUC) for visual features.

and videos. Moreover, the CD features that are significantly faster to extract have comparable performance to the WSC features that require training expensive SVMs. Finally, the WSC and CD features detected using DT are stronger than the ones using D-SIFT.

Feature	MAP	AUC
$\text{WSC}^{\text{MFCC}}_{\text{TRECVID}}$	0.76%	0.507
CD^{MFCC}	**1.04%**	**0.604**

Table 5.3: Comparison of mean average precision (MAP) and area under the curve (AUC) for audio features.

5.6.3 Comparison of Audio Features

We compare the performance of our WSC and CD features trained using the audio MFCC features. Table 5.3 summarizes the MAP and AUC results. As observed, both of the audio features are weaker than the visual features.

5.6.4 Comparison of Language Features

Feature	MAP	AUC
ASR	3.66%	0.583
OCR (word)	4.30%	**0.636**
OCR (character)	**4.72%**	0.611

Table 5.4: Comparison of mean average precision (MAP) and area under the curve (AUC) for language features.

Table 5.4 compares the performance of our OCR and ASR systems. All the systems have higher MAP compared to the visual and audio features from Tables 5.2 and 5.3. However, note that the AUCs of many visual features outperform the language features. This is because although language content, when present, is a highly accurate source of information, its occurrence is sporadic, leading to low recall.

5.6.5 Comparison of Fusion Systems

Feature	MAP	AUC
ASR	3.66%	0.583
OCR	5.87%	0.642
Audio	1.04%	0.623
Visual (CD + WSC)	6.12%	**0.853**
Full	**12.65**%	0.733

Table 5.5: Comparison of mean average precision (MAP) and area under the curve (AUC) for fusion systems.

We fused each of the individual systems described above, both within each modality as well as across modalities. Table 5.5 compares the performance of the different fusion systems. Note that within the visual system, we found that off-the-shelf visual features did not improve the fused system, and only included our CD and

WSC features. While none of the individual visual features is stronger than ASR or OCR, the visual system is the single strongest system after fusion, gaining ∼75% relative improvement over the single best visual system. The combined OCR system also outperforms the individual OCR systems, and the full system that combines all modalities more than doubles the performance of any individual modality as measured by MAP.

5.6.6 TRECVID Performance

The zero-shot event detection task was introduced as a pilot training condition as part of the TRECVID MED 13 evaluations. Independent evaluations were conducted by NIST on a blind ∼100000 video dataset, both for the same 20 events as in our previous experiments (*prespecified*), as well as for 10 new events given one week before the evaluation (*ad hoc*). Our zero-shot system achieved highly competitive scores for both prespecified and ad hoc conditions, placing among the top three out of 9 submissions. In particular, our consistent performance between prespecified and ad hoc events demonstrate the robustness of our event-independent approach to generalize to new queries.

5.7 Discussion and Conclusion

Only limited attention has been devoted to the task of video retrieval using only text queries. We present a systematic evaluation of our zero-shot framework for performing high-level multimedia event detection with no training data, given

only text descriptions of the events of interest. Our findings and results on the large TRECVID MED dataset can serve as an initial baseline for this challenging task.

We present a general framework for zero-shot learning, that utilizes multiple multi-modal features to map a video to an intermediate semantic attribute space, which are then projected to a high-dimensional concept space using statistics learned on a large text corpus. Similarity between the attributes and a text query are computed in this space, and the scores computed from different attribute sets are combined to get the final score. We demonstrate the effectiveness of this approach for aligning disjoint vocabularies between query and various modalities.

We describe two simple but effective methods for rapidly training new concept detectors using in-domain as well as web data in the form of image/video with associated text descriptions. Detailed experimental results show that our concept detectors significantly outperform off-the-shelf detectors for zero-shot retrieval tasks. Exploiting the complementary nature of speech and video text as well as between different concept banks, we perform multiple rounds of fusion to produce a final system that is significantly better than any individual feature or modality.

Bibliography

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[2] Geoffrey Hinton Laurens van der Maaten. Visualizing data using t-sne. In *Journal of Machine Learning Research, Vol. 9*, pages pp. 2579–2605. 2008.

[3] Paul Over, George Awad, Martial Michel, Jonathan Fiscus, Greg Sanders, Wessel Kraaij, Alan F. Smeaton, and Georges Quéenot. Trecvid 2013 – an overview of the goals, tasks, data, evaluation mechanisms and metrics. In *Proceedings of TRECVID 2013*. NIST, USA, 2013.

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

[5] Li-Jia Li, Hao Su, Eric Xing, and Li Fei-Fei. Object bank: A high-level image representation for scene classification and semantic feature sparsification. In *NIPS*, 2010.

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

[7] Wei Liu, Jun Wang 0006, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. In *CVPR*, 2012.

[8] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton van den Hengel, and David Suter. Fast supervised hashing with decision trees for high-dimensional data. In *CVPR' 14*.

[9] Tiezheng Ge, Kaiming He, and Jian Sun. Graph cuts for supervised binary coding. In *ECCV '14*.

[10] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. Supervised discrete hashing. *CVPR' 15*.

[11] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions.

[12] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *NIPS, 2009*.

[13] Wei Liu, Jun Wang, and Shih fu Chang. Hashing with graphs. In *In ICML*, 2011.

[14] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. Discrete graph hashing. In *NIPS*, 2014.

[15] Mohammad Norouzi and David J. Fleet. Minimal loss hashing for compact binary codes. In *ICML'11*.

[16] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, 2014.

[17] Guosheng Lin, Chunhua Shen, David Suter, and Anton van den Hengel. A general two-step approach to learning-based hashing. In *ICCV' 13*.

[18] Mohammad Rastegari, Ali Farhadi, and David A. Forsyth. Attribute discovery via predictable discriminative binary codes. In *ECCV '12*.

[19] Qifan Wang, Bin Shen, Shumiao Wang, Liang Li, and Luo Si. Binary codes embedding for fast image tagging with incomplete labels. In *ECCV '14*.

[20] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS' 09*.

[21] Jonathan Masci, Alexander M. Bronstein, Michael M. Bronstein, Pablo Sprechmann, and Guillermo Sapiro. Sparse similarity-preserving hashing. 2014.

[22] Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou. Deep hashing for compact binary codes learning. In *CVPR*, 2015.

[23] Fang Zhao, Yongzhen Huang, Liang Wang, and Tieniu Tan. Deep semantic ranking based hashing for multi-label image retrieval. *CVPR*, 2015.

[24] Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, 2015.

[25] Kevin Lin, Huei-Fang Yang, Jen-Hao Hsiao, and Chu-Song Chen. Deep learning of binary hash codes for fast image retrieval. In *CVPR Workshops*, 2015.

[26] Weiran Wang and Miguel Á. Carreira-Perpiñán. The role of dimensionality reduction in classification. In *AAAI' 14*.

[27] Liang Sun, Shuiwang Ji, and Jieping Ye. Canonical Correlation Analysis for Multilabel Classification: A Least-Squares Formulation, Extensions, and Analysis. *IEEE*, 2011.

[28] Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR'11*.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*. 2012.

[30] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.

[31] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[32] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

[33] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM MM '14*.

[34] Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE' 08*.

[35] Google search by image. http://www.google.com/insidesearch/features/images/searchbyimage.html.

[36] Tineye : Reverse image search. https://www.tineye.com/.

[37] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE International Conference on Computer Vision (ICCV)*, 2009.

[38] Kilian Q. Weinberger and Olivier Chapelle. Large margin taxonomy embedding for document categorization. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1737–1744. Curran Associates, Inc., 2009.

[39] 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.

[40] Jorge Sanchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. Image classification with the fisher vector: Theory and practice. 2013.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, page 2012.

[42] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[43] Genevieve Patterson, Chen Xu, Hang Su, and James Hays. The sun attribute database: Beyond categories for deeper scene understanding. *International Journal of Computer Vision*, 108(1-2):59–81, 2014.

[44] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[45] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[46] Ning Zhang, Jeff Donahue, Ross Girshick, and Trevor Darrell. Part-based R-CNNs for fine-grained category detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.

[47] David R. Hardoon, Sandor Szedmak, Or Szedmak, and John Shawe-taylor. Canonical correlation analysis; an overview with application to learning methods. Technical report, 2007.

[48] Chun-Nam Yu and T. Joachims. Learning structural svms with latent variables. In *International Conference on Machine Learning (ICML)*, 2009.

[49] Lei Zhang, Yongdong Zhang, Jinhui Tang, Xiaoguang Gu, Jintao Li, and Qi Tian. Topology preserving hashing for similarity search. In Alejandro Jaimes, Nicu Sebe, Nozha Boujemaa, Daniel Gatica-Perez, David A. Shamma, Marcel Worring, and Roger Zimmermann, editors, *ACM Multimedia*, pages 123–132. ACM, 2013.

[50] Daniel Hsu, Sham Kakade, John Langford, and Tong Zhang. Multi-label prediction via compressed sensing. In Yoshua Bengio, Dale Schuurmans, John D. Lafferty, Christopher K. I. Williams, and Aron Culotta, editors, *NIPS*, pages 772–780. Curran Associates, Inc., 2009.

[51] Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *CVPR*, pages 951–958. IEEE, 2009.

[52] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6:1453–1484, December 2005.

[53] Jason Weston, Samy Bengio, and Nicolas Usunier. N.: Wsabie: Scaling up to large vocabulary image annotation. *In: IJCAI*, pages 2764–2770.

[54] Zeynep Akata, Florent Perronnin, Zaid Harchaoui, and Cordelia Schmid. Attribute-Based Classification with Label-Embedding. In *NIPS 2013 Workshop on Output Representation Learning*, Lake Tahoe, United States, December 2013. Neural Information Processing Systems (NIPS) Foundation.

[55] Jia Deng, Alexander C. Berg, Kai Li, and Li Fei-Fei. What does classifying more than 10,000 image categories tell us? In *Proceedings of the 11th European Conference on Computer Vision: Part V*, ECCV'10, pages 71–84, Berlin, Heidelberg, 2010. Springer-Verlag.

[56] Andrea Frome, Gregory S. Corrado, Jonathon Shlens, Samy Bengio, Jeffrey Dean, Marc'Aurelio Ranzato, and Tomas Mikolov. Devise: A deep visual-semantic embedding model. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *NIPS*, pages 2121–2129, 2013.

[57] Thomas Mensink, Efstratios Gavves, and Cees G. M. Snoek. Costa: Co-occurrence statistics for zero-shot classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, Ohio, USA, June 2014.

[58] Yair Weiss, Rob Fergus, and Antonio Torralba. Multidimensional spectral hashing. In *ECCV (5)*, pages 340–353, 2012.

[59] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, San Francisco, USA, June 2010.

[60] Yurii Nesterov. *Introductory lectures on convex optimization : a basic course*. Applied optimization. Kluwer Academic Publ., Boston, Dordrecht, London, 2004.

[61] Tom Goldstein, Christoph Studer, and Richard G. Baraniuk. FASTA: A generalized implementation of forward-backward splitting. *CoRR*, abs/1501.04979, 2015.

[62] Roman Rosipal and Nicole Krämer. Overview and Recent Advances in Partial Least Squares. In Craig Saunders, Marko Grobelnik, Steve Gunn, and John Shawe-Taylor, editors, *Subspace, Latent Structure and Feature Selection*, volume 3940 of *Lecture Notes in Computer Science*, chapter 2, pages 34–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[63] Florent Perronnin and Christopher R. Dance. Fisher kernels on visual vocabularies for image categorization. In *CVPR*, 2007.

[64] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob J. Verbeek. Image classification with the fisher vector: Theory and practice. *International Journal of Computer Vision*, 105(3):222–245, 2013.

[65] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *IEEE Conference on Computer Vision & Pattern Recognition*, pages 3304–3311, jun 2010.

[66] Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas S. Huang, and Yihong Gong. Locality-constrained linear coding for image classification. In *CVPR*, pages 3360–3367, 2010.

[67] Florent Perronnin, Yan Liu, Jorge Sánchez, and Herve Poirier. Large-scale image retrieval with compressed fisher vectors. In *CVPR*, pages 3384–3391, 2010.

[68] Jorge Sánchez and Florent Perronnin. High-dimensional signature compression for large-scale image classification. In *CVPR*, pages 1665–1672, 2011.

[69] Luca Marchesotti, Claudio Cifarelli, and Gabriela Csurka. A framework for visual saliency detection with applications to image thumbnailing. In *ICCV*, pages 2232–2239. IEEE, 2009.

[70] 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. to appear.

[71] Mohammad Norouzi and David Fleet. Cartesian k-means. In *IEEE Conference on Computer Vision & Pattern Recognition*, 2013.

[72] Youngwoon Lee. Spherical hashing. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 2957–2964, Washington, DC, USA, 2012. IEEE Computer Society.

[73] Yunchao Gong, Sanjiv Kumar, Henry Rowley, and Svetlana Lazebnik. Learning binary codes for high dimensional data using bilinear projections. In *IEEE Computer Vision and Pattern Recognition*, 2013.

[74] F. X. Yu, S. Kumar, Y. Gong, and S.-F. Chang. Circulant Binary Embedding. *ArXiv e-prints*, May 2014.

[75] A. Joly and O. Buisson. Random maximum margin hashing. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 873–880, Washington, DC, USA, 2011. IEEE Computer Society.

[76] Junfeng He, Regunathan Radhakrishnan, Shih-Fu Chang, and Claus Bauer. Compact hashing with joint optimization of search accuracy and time. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), oral session*, June 2011.

[77] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, pages 94–105, New York, NY, USA, 1998. ACM.

[78] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *Proceedings of the 10th European Conference on Computer Vision: Part I*, ECCV '08, pages 304–317, Berlin, Heidelberg, 2008. Springer-Verlag.

[79] Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cedric Bray. Visual categorization with bags of keypoints. In *ECCVW*, 2004.

[80] David G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60:91–110, 2004.

[81] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *CVIU*, 110(3):346–359, 2008.

[82] K. van de Sande, T. Gevers, and C. Snoek. Evaluating color descriptors for object and scene recognition. 32(9):1582–1596, 2010.

[83] Ivan Laptev. On space-time interest points. *IJCV*, 64(2-3):107–123, 2005.

[84] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Dense trajectories and motion boundary descriptors for action recognition. *IJCV*, 103(1):60–79, 2013.

[85] Mark Palatucci, Dean Pomerleau, Geoffrey Hinton, and Tom Mitchell. Zero-shot learning with semantic output codes. In *NIPS*, December 2009.

[86] R. Arandjelović and A. Zisserman. Multiple queries for large scale specific object retrieval. In *BMVC*, 2012.

[87] David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. English gigaword third edition. In *Linguistic Data Consortium, Philadelphia*, 2007.

[88] Jan C. van Gemert, Cor J. Veenman, Arnold W. M. Smeulders, and Jan-Mark Geusebroek. Visual word ambiguity. *IEEE PAMI*, 32(7):1271–1283, 2010.

[89] Y.L. Boureau, F. Bach, Y.L. Le Cun, and J. Ponce. Learning mid-level features for recognition. In *CVPR*, 2010.

[90] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob J. Verbeek. Image classification with the fisher vector: Theory and practice. *International Journal of Computer Vision*, 105(3):222–245, 2013.

[91] Olga Russakovsky and Li Fei-Fei. Attribute learning in large-scale datasets. In *ECCV*, Crete, Greece, September 2010.

[92] Marcus Rohrbach, Michael Stark, and Bernt Schiele. Evaluating knowledge transfer and zero-shot learning in a large-scale setting. In *CVPR*, 2011.

[93] Pichai Kankuekul, Aram Kawewong, Sirinart Tangruamsub, and Osamu Hasegawa. Online incremental attribute-based zero-shot learning. In *CVPR*, pages 3657–3664, 2012.

[94] B. Hariharan, S. V. N. Vishwanathan, and M. Varma. Efficient max-margin multi-label classification with applications to zero-shot learning. *Machine Learning Journal*, 88(1):127–155, 2012.

[95] Mohamed Elhoseiny, Babak Saleh, and Ahmed Elgammal. Write a classifier: Zero-shot learning using purely textual descriptions. In *ICCV*, 2013.

[96] Jeffrey Dalton, James Allan, and Mirajkar Pranav. Zero-shot video retrieval using content and concepts. In *ACM Conference of Information and Knowledge Management*, 2013.

[97] Yusuf Aytar, Mubarak Shah, and Jiebo Luo. Utilizing semantic word similarity measures for video retrieval. In *CVPR*, pages 1–8. IEEE, 2008.

[98] Min Young Jung and Sung Han Park. Semantic similarity based video retrieval. In *New Directions in Intelligent Interactive Multimedia Systems and Services-2*, pages 381–390. Springer, 2009.

[99] Kobus Barnard, Pinar Duygulu, and David A. Forsyth. Clustering art. In *CVPR*, pages 434–441, 2001.

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

[101] Girish Kulkarni, Visruth Premraj, Sagnik Dhar, Siming Li, Yejin Choi, Alexander C. Berg, and Tamara L. Berg. Baby talk: Understanding and generating simple image descriptions. In *CVPR*, pages 1601–1608, 2011.

[102] Yezhou Yang, Ching Lik Teo, Hal Daumé III, and Yiannis Aloimonos. Corpus-guided sentence generation of natural images. In *EMNLP*, pages 444–454, 2011.

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

[104] Niveda Krishnamoorthy, Girish Malkarnenkar, Raymond J. Mooney, Kate Saenko, and Sergio Guadarrama. Generating natural-language video descriptions using text-mined knowledge. pages 541–547, 2013.

[105] P. Das, C. Xu, R. F. Doell, and J. J. Corso. A thousand frames in just a few words: Lingual description of videos through latent topics and sparse object stitching. In *CVPR*, 2013.

[106] Jia Deng, Alexander C. Berg, Kai Li, and Li Fei-Fei. What does classifying more than 10,000 image categories tell us? In *ECCV*, pages 71–84, 2010.

[107] Richard Socher, Milind Ganjoo, Hamsa Sridhar, Osbert Bastani, Christopher D. Manning, and Andrew Y. Ng. Zero-shot learning through cross-modal transfer. *CoRR*, abs/1301.3666, 2013.

[108] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. In *IEEE ASSP*, volume 28, pages 357–66, 1980.

[109] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *ECCV*, volume 6314 of *Lecture Notes in Computer Science*, pages 143–156. Springer Berlin Heidelberg, 2010.

[110] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *ICCV*, 2007.

[111] Paul Over, George Awad, Martial Michel, Jonathan Fiscus, Wessel Kraaij, Alan F. Smeaton, and Georges Quéenot. TrecVid 2011 – An Overview of the Goals, Tasks, Data, Evaluation Mechanisms and Metrics. In *Proceedings of TRECVID 2011*. NIST, USA, 2011.

[112] Pradeep Natarajan, Shuang Wu, Shiv Naga Prasad Vitaladevuni, Xiaodan Zhuang, Stavros Tsakalidis, Unsang Park, Rohit Prasad, and Premkumar Natarajan. Multimodal feature fusion for robust event detection in web videos. In *CVPR*, 2012.