Chapter 6

Learning the Hashing Hypersurfaces

The research presented in this Chapter has been previously published in Moran and Lavrenko (2015a) and Moran and Lavrenko (2015b).

6.1 Introduction

In Chapters 1-2 I previously discussed how the generation of similarity preserving hashcodes involves two main steps carried out sequentially: low-dimensional projection followed by binary quantisation. In Chapters 4 and 5 I then introduced two novel data-driven quantisation algorithms for converting real-valued projections into binary hashcodes. In most cases both of these data-driven models were shown to achieve a significantly higher retrieval effectiveness than the commonly used data-independent single bit quantisation (SBQ) algorithm and a host of more recently proposed data-dependent quantisation algorithms. Having argued the validity of this dissertation's thesis for the quantisation operation I will now turn my attention in this chapter to the low-dimensional projection function.

Recall from my review of previous related research in Chapter 2 how Locality Sensitive Hashing (LSH), a seminal algorithm for solving the ANN search problem, partitions the input feature space with K randomly drawn hyperplanes which are selected independent of the distribution of the data. Data-points are projected onto the normal vectors to these K hyperplanes and the subsequent real-valued projections quantised to form a K-bit hashcode. The central argument in this chapter is that this randomised projection leads to a sub-optimal space partitioning. By relaxing assumption A_3 outlined in Chapter 1, I hypothesise that a significantly higher retrieval effectiveness can be achieved by learning task-specific hashing hypersurfaces. This hypothesis is inspired

by previous research in the learning to hash literature which has presented convincing evidence as to the benefits of inducing a data-dependence into the low-dimensional hashing projection function. These recently proposed data-dependent projection functions were reviewed in detail in Chapter 2, Sections 2.6.3-2.6.4 and will form a strong set of baselines for the novel contributions I make in this chapter. I introduce a new supervised projection function dubbed *Graph Regularised Hashing (GRH)* which operations in three steps. In the first step the proposed projection function leverages the principle of *graph regularisation* (Diaz (2007)) which smooths the distribution of binary bits so that neighbouring data-points, as indicated by the adjacency graph, are much more likely to be assigned identical bits than non-neighbouring data-points. This concept is reminiscent of the well-known *Cluster Hypothesis* of Information Retrieval (IR) which states that *"closely associated documents tend to be relevant to the same requests"* (Rijsbergen (1979)). In the second step the regularised bits are then used as targets for a set of binary classifiers that separate opposing bits with maximum margin. In the final step the training data-points are relabeled with updated hashcodes using the learnt hyperplanes. Iterating these three steps permits the hashing hypersurfaces to evolve into positions within the input feature space that better separate opposing bits versus a purely random LSH partitioning. I show in Figure 6.1 a t-SNE visualisation (van der Maaten and Hinton (2008)) of an embedding creating by my proposed algorithm on the CIFAR-10 dataset (Krizhevsky and Hinton (2009)). This diagram illustrates how the proposed supervised projection function is able to group together related images within the projected space.

In one of the foremost contributions of this thesis I further show how to extend the proposed projection function to learn hashing hypersurfaces that can generate effective hashcodes for similar data-points existing in *two different feature spaces*, for example an image and a document that both describe a tiger in a jungle setting. This latter contribution effectively brings the computational advantages of hashing-based ANN search to the plethora of multi-modal datasets in existence in the modern data rich world, datasets which are entirely out-of-reach for conventional unimodal projection functions such as LSH.

In the experimental evaluation I demonstrate a host of new and previously unexpected results arising from examining the effectiveness and efficiency of this supervised projection function. Most notably the projection function is able to out-perform a large swath of state-of-the-art supervised and unsupervised data-dependent projection functions using linear hypersurfaces (hyperplanes) and without the need to solve

Method	Data-Dependent	Supervised	Scalable	Effectiveness
LSH			✓	Low
SH	✓			Low
STH	✓	✓		Medium
BRE	✓	✓		Medium
ITQ+CCA	✓	✓		Medium
KSH	✓	✓		High
GRH	✓	✓	✓	**High**

Table 6.1: Comparison of the projection function introduced in this chapter (GRH) versus the most closely related projections functions from the literature. All of the baselines were previously reviewed in Chapter 2.

a computationally expensive eigenvalue problem. This property is one of the main bottlenecks that severely hamper the scalability of many previously proposed data-dependent projection functions for hashing. The properties of my proposed algorithm (GRH) as rated against closely related research along the four dimensions of *data-dependence*, *supervision*, *scalability* and *effectiveness* is given in Table 6.1. To the best of my knowledge the research presented in this chapter introduces one of the first known supervised projection functions that has the desirable properties of being both effective *and* scalable, a true rarity in the literature.

The remainder of this chapter is structured in the following manner: in Section 6.2 I give an overview of the problem definition and then introduce my unimodal graph regularised projection function that integrates graph regularisation into a multi-step iterative hash function learning framework. The algorithm overview is subsequently followed in Section 6.3 with a comprehensive set of experiments designed to measure both the retrieval effectiveness and efficiency of the projection function on the now familiar task of query-by-example image retrieval. I conclude this first part of the chapter in Section 6.4 with a summary of the contributions and a discussion on the main experimental findings. I then show how to extend this unimodal model to cross-modal hashing in Section 6.5. This is followed in Section 6.6 by a set of experiments designed to compare the model to prior-art. The chapter is concluded in Section 6.8 by summarising the main contributions.

Figure 6.1: t-SNE visualisation of a subset of the CIFAR-10 image dataset (20,000 images). Image positions are computed using a 2-dimensional t-SNE projection (van der Maaten and Hinton (2008)) of a 32-dimensional embedding produced by the non-linear variant of my graph regularised projection function. Clusters of semantically related images are readily evident such as horses (bottom), airplanes (top right) and dogs (top left). Image best viewed in colour and ideally with a zoom tool on the electronic PDF version of the thesis.

6.2 A Graph Regularised Projection Function

6.2.1 Problem definition

The problem definition in this chapter is similar to the definitions given in Chapter 4 and Chapter 5 but with the emphasis now placed on learning the *K hashing hyperplanes* $\{\mathbf{h}_k \in \mathbb{R}^D\}_{k=1}^K$ with normal vectors $\{\mathbf{w}_k \in \mathbb{R}^D\}_{k=1}^K$ rather than on the setting of the quantisation thresholds. To recapitulate the problem setup: we are given a dataset of N points $\mathbf{X} = [\mathbf{x}_1 \ldots \mathbf{x}_N]^\intercal$, where each point $\mathbf{x}_i$ is a D-dimensional vector of real-valued features. My goal again is to represent each item with a binary hashcode $\mathbf{b}_i \in \{0,1\}^K$ consisting of K bits. In this chapter my aim is to learn the hashing hyperplanes in a way that the hashcodes $\mathbf{b}_i, \mathbf{b}_j$ generated by those hyperplanes will be have a low Hamming distance for neighbouring points $\mathbf{x}_i, \mathbf{x}_j$. As before, the neighbourhood structure between the data-points in the input feature space is encoded in a binary adjacency matrix $\mathbf{S}$, where $S_{ij} = 1$ if points $\mathbf{x}_i$ and $\mathbf{x}_j$ are considered neighbours, and $S_{ij} = 0$ otherwise.

6.2.2 Overview of the approach

My proposed projection function iteratively performs three steps: *(A) regularisation*, where we make the N_{trd} hashcodes $\{\mathbf{b}_1 \ldots \mathbf{b}_{N_{trd}}\}$ more consistent with the adjacency matrix $\mathbf{S} \in \mathbb{R}^{N_{trd} \times N_{trd}}$; *(B) partitioning*, where we learn a set of hypersurfaces $\{\mathbf{h}_1 \ldots \mathbf{h}_K\}$ that subdivide the space $\mathbb{R}^D$ into regions that are consistent with the hashcodes. These hypersurfaces are needed to efficiently compute the hashcodes for testing points $\mathbf{x} \in \mathbb{R}^D$, where we have no affinity information available. *(C) Prediction*, in which the learnt hyperplanes are used to generate updated hashcodes for the training data-points. I initialise the hashcodes $\{\mathbf{b}_1 \ldots \mathbf{b}_{N_{trd}}\}$ by running the points $\{\mathbf{x}_1 \ldots \mathbf{x}_{N_{trd}}\}$ through any existing unsupervised or supervised projection function, such as LSH (Indyk and Motwani (1998)) or ITQ+CCA (Gong and Lazebnik (2011))[1]. I then iterate the three steps in a manner reminiscent of the *EM algorithm* (Dempster et al. (1977)): the regularised hashcodes from step A adjust the hypersurfaces in step B, and these surfaces in turn generate new hashcodes in Step C which are then passed into step A. The algorithm is run for a fixed number of iterations (M). Further details on Steps A,B,C of the algorithm are provided in Sections 6.2.3-6.2.5.

[1] LSH and ITQ+CCA were reviewed in Chapter 2, Section 2.4 and 2.6.4.1, respectively.

6.2.3 Step A: Regularisation

I take a graph-based approach to regularising the hashcodes. The nodes of the graph correspond to the points $\{\mathbf{x}_1 \ldots \mathbf{x}_{N_{trd}}\}$ and $\mathbf{S}$ plays the role of an adjacency matrix: I insert an undirected edge between nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ if and only if $S_{ij} = 1$. Each node $\mathbf{x}_i$ is annotated with K binary labels, corresponding to the K bits of the hashcode $\mathbf{b}_i$. Our aim is to increase the similarity of the label sets at the opposite ends of each edge in the graph. I achieve this by averaging the label set of each node with the label sets of its immediate neighbours. This is similar to the *score regularisation* method of Diaz (2007) and the *label propagation* algorithm of Zhu and Ghahramani (2002), although my update equation is slightly different.

Figure 6.2 illustrates my approach. On the top I show a graph with 8 nodes $\{a \ldots h\}$ and edges showing the nearest-neighbour constraints. Each node is annotated with 3 labels which reflect the initial hashcode of the node (zero bits are converted to labels of -1). On the bottom of Figure 6.2 I show the effect of label propagation for nodes c and e (which are immediate neighbours). Node e has initial labels $[+1,-1,-1]$ and 3 neighbours with the following label sets: c:$[+1,+1,+1]$, f:$[+1,+1,+1]$ and g:$[+1,+1,-1]$. I aggregate these four sets and look at the sign of the result to obtain a new set of labels for node e: $\text{sgn}[\frac{+1+1+1+1}{4}, \frac{-1+1+1+1}{4}, \frac{-1+1+1-1}{4}] = [+1,+1,-1]$. Note that the second label of e has become more similar to the labels of its immediate neighbours. Formally, I regularise the labels via the following equation:

$$\mathbf{B}_m \leftarrow \text{sgn}\left(\alpha\, \mathbf{S}\mathbf{D}^{-1}\mathbf{B}_{m-1} + (1-\alpha)\mathbf{B}_0\right) \tag{6.1}$$

This equation effectively *diffuses* bits over the image-image similarity graph $\mathbf{S}$. Here $m \in [1,\ldots,M]$, where M is the maximum number of iterations, $\mathbf{S}$ is the adjacency matrix and $\mathbf{D}$ is a diagonal matrix containing the degree of each node in the graph[2]. $\mathbf{B} \in \{-1,+1\}^{N_{trd} \times K}$ represents the labels assigned to every node at the previous step of the algorithm, $\mathbf{B}_0$ indicates the labels at iteration 0, namely as initialised by LSH or ITQ+CCA, $\alpha \in [0,1]$ is a scalar smoothing parameter and sgn represents the sign function, modified so that $\text{sgn}(0) = -1$. The hashcodes at the current iteration $\mathbf{B}_m$ are set to be a convex combination of the hashcodes at the previous iteration $\mathbf{B}_{m-1}$ and the initialised hashcodes ($\mathbf{B}_0$).

[2] $\mathbf{D}^{-1}$ has the effect of L_1-normalising the rows of $\mathbf{S}$.

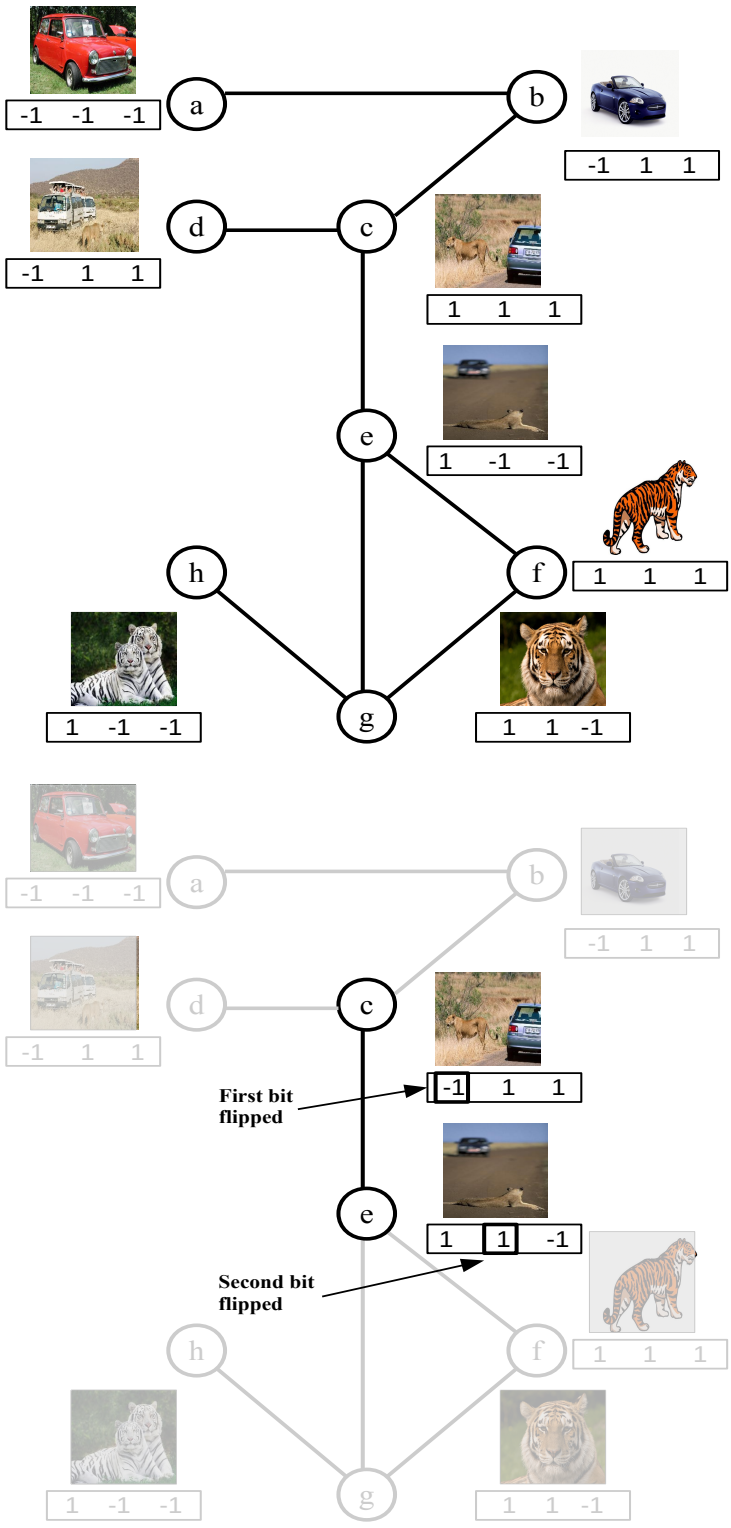


Figure 6.2: The regularisation step. Nodes represent data-points and arcs represent neighbour relationships. The 3-bit hashcode assigned to a given node is shown in the boxes. Top: The original hashcode assignment at initialisation. Bottom: The hashcode update for nodes c and e.

6.2.4 Step B: Partitioning

At the end of step A, each point $\mathbf{x}_i$ has K binary labels $\{-1,+1\}$. I use these labels to learn a set of hypersurfaces $\{\mathbf{h}_1 \ldots \mathbf{h}_K\}$. Each surface $\mathbf{h}_k \in \mathbb{R}^D$ will partition the space $\mathbb{R}^D$ into two disjoint regions: *positive* and *negative*. The positive region of $\mathbf{h}_k$ should envelop all points $\mathbf{x}_i$ for which the k'th label was $+1$; while the negative region should contain all the $\mathbf{x}_i$ for which $B_{ik} = -1$. For simplicity, I restrict the discussion to linear hypersurfaces (hyperplanes) in this section, but a non-linear generalisation is straightforward via the kernel trick (Bishop (2006)). In particular, I discuss how my model can be transformed into a non-linear hash function in Section 6.3.3.8, and I compare the performance of linear and non-linear boundaries in Section 6.3.

A hyperplane is defined by the normal vector $\mathbf{w}_k \in \mathbb{R}^D$ and a scalar bias $t_k \in \mathbb{R}$. Its positive region consists of all points $\mathbf{x}$ for which $\mathbf{w}_k^\intercal \mathbf{x} + t_k > 0$. I position each hyperplane $\mathbf{h}_k$ to maximise the margin, i.e. the separation between the points $\mathbf{x}_i$ that have $B_{ik} = -1$ and those that have $B_{ik} = +1$. I find the maximum-margin hyperplanes by independently solving K constrained optimisation problems:

$$\begin{aligned} \text{for } k = 1 \ldots K: \quad & \min \ ||\mathbf{w}_k||^2 + C \textstyle\sum_{i=1}^{N_{trd}} \xi_{i,k} \\ \text{s.t.} \quad & B_{ik}(\mathbf{w}_k{}^\intercal \mathbf{x}_i + t_k) \geq 1 - \xi_{i,k} \quad \text{for } i = 1 \ldots N_{trd} \end{aligned} \tag{6.2}$$

Here $\xi_{ik} > 0$ are slack variables that allow some points $\mathbf{x}_i$ to fall on the wrong side of the hyperplane $\mathbf{h}_k$; and $C \in \mathbb{R}_+$ is a parameter that allows us to trade off the size of the margin $\frac{1}{||\mathbf{w}_k||}$ against the number of points misclassified by $\mathbf{h}_k$. I solve the optimisation problem in equation (6.2) using `liblinear` Fan et al. (2008) and `libSVM` Chang and Lin (2011) for linear and non-linear hypersurfaces respectively.

Figure 6.3 illustrates step B for linear hypersurfaces. On the top I show the hyperplane $\mathbf{h}_1$ that partitions the points $a \ldots h$ using their first label as the target. Nodes a, b, c, d have the first label set to -1, while e, f, g, h are labelled as $+1$. The hyperplane $\mathbf{h}_1$ is a horizontal line, equidistant from points c and e: this provides maximum possible separation between the positives and the negatives. No points are misclassified, so all the slack variables $\xi_{i,1}$ are zero. The bottom of Figure 6.3 shows the maximum-margin hyperplane $\mathbf{h}_2$ that partitions the points based on their second label. In this case, perfect separation is not possible, and $\xi_{i,2}$ is non-zero (nodes g and d are on the wrong side of $\mathbf{h}_2$).

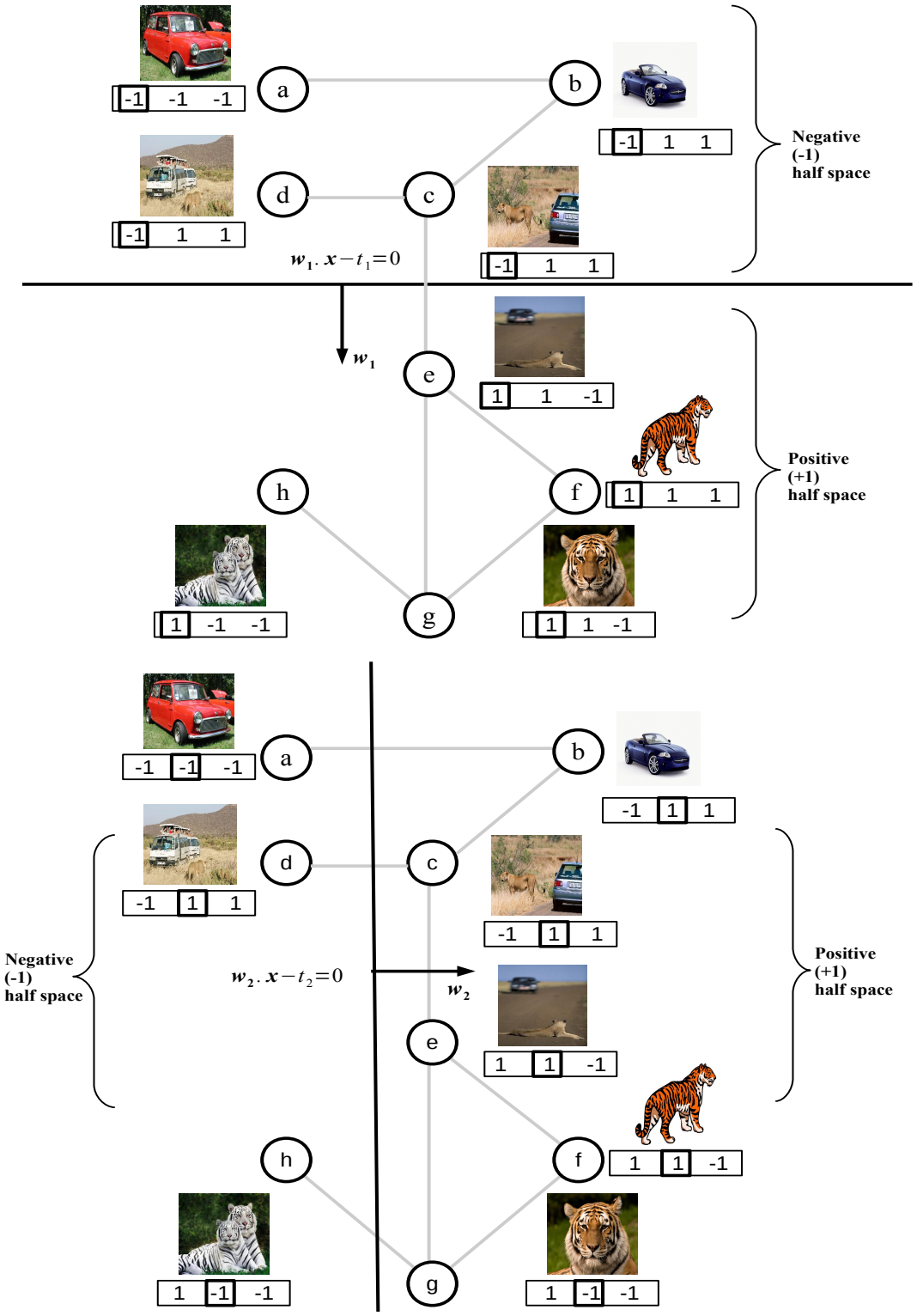


Figure 6.3: The partitioning step. In this stage, the regularised hashcodes are used to re-position the hashing hyperplanes. Top: First bit of hashcode. Bottom: Second bit.

6.2.5 Step C: Prediction

In the third step the estimated hyperplane normal vectors $\{\mathbf{w}_1 \ldots \mathbf{w}_K\}$ are used to re-label the data-points:

$$B_{ik} = \text{sgn}(\mathbf{w}_k{}^\intercal \mathbf{x}_i + t_k) \text{ for } i = \{1 \ldots N_{trd}\} \text{ and } k = \{1 \ldots K\} \tag{6.3}$$

The effect of this step is that points which could not be classified correctly will now be relabelled to make them consistent with all hyperplanes. For example, the second label of node g in Figure 6.3 will change from -1 to $+1$ to be consistent with $\mathbf{h}_2$. These new labels are passed back into step A for the next iteration of the algorithm. After the last iteration, I use the hyperplane normal vectors $\{\mathbf{w}_1 \ldots \mathbf{w}_K\}$ to predict hashcodes for new instances $\mathbf{x}$: the k'th bit in the code is set to 1 if $\mathbf{w}_k^\intercal \mathbf{x} + t_k > 0$, otherwise it is zero.

6.2.6 Algorithm Specification

Algorithm 9 presents the pseudo-code for my graph regularised projection function. In Line 1 the hashcodes for the N_{trd} training data-points are initialised in matrix $\mathbf{B}$ with an existing projection function such as LSH or ITQ+CCA. In Line 2, hashcode bits that are 0 are converted to -1. The main loop of the algorithm is shown in Line 4 which is repeated for M iterations. Line 5 is the regularisation step in which hashcodes in $\mathbf{B}$ are made more similar to their neighbours as specified by the affinity matrix $\mathbf{S}$. Line 6 is the start of the loop that learns the K new hyperplanes for each bit position, by training K SVM classifiers using the regularised bits in $\mathbf{B}$ as training labels (Line 8). The learnt hyperplanes are used to update the hashcode bits in Line 11. At the end of the M iterations, the learnt hyperplanes $\{\mathbf{w}_k \in \mathbb{R}^D\}_{k=1}^K$, $\{t_k \in \mathbb{R}\}_{k=1}^K$ can be used to generate the hashcodes for novel data-points (Line 13). Figure 6.4 illustrates the operation of GRH on a synthetic dataset consisting of three clusters

6.2.7 Computational Complexity

If we let N_{trd} denote the number of training data-points then the graph regularisation step is of $O(N_{trd}^2 K)$. Training a linear SVM takes $O(N_{trd} DK)$ time (Joachims (2006)) while prediction (test time) is $O(N_{trd} DK)$. Therefore linear GRH is $O(MN_{trd}^2 K)$ for M iterations. Typically the adjacency matrix $\mathbf{S}$ is sparse[3], $N_{trd} \ll N$ and K is small (≤ 128 bits) thereby ensuring that the *linear* variant of GRH is scalable to large datasets. The

[3]For example, around 10% of the entries in $\mathbf{S}$ are non-zero for the CIFAR-10 dataset.

Algorithm 9: GRAPH REGULARISED HASHING (GRH)

Input: Dataset $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D}$, adjacency matrix $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$, degree matrix $\mathbf{D} \in \mathbb{Z}_+$, interpolation parameter $\alpha \in [0,1]$, number of iterations $M \in \mathbb{Z}_+$

Output: Hyperplanes $\{\mathbf{w}_1 \ldots \mathbf{w}_K\}$, biases $\{t_1 \ldots t_K\}$

Initialise $\mathbf{B}_0 \in \{0,1\}^{N_{trd} \times K}$ via LSH or ITQ+CCA from $\mathbf{X}$
$\mathbf{B}_0 = \text{sgn}(\mathbf{B}_0 - \frac{1}{2})$
$\mathbf{B} = \mathbf{B}_0$
for $m \leftarrow 1$ ***to*** M **do**
    $\mathbf{B} = \text{sgn}\left(\alpha \mathbf{S} \mathbf{D}^{-1} \mathbf{B} + (1-\alpha)\mathbf{B}_0\right)$
    for $k \leftarrow 1$ ***to*** K **do**
        $\mathbf{b}_k = \mathbf{B}(:,k)$
        Train SVM_k with $\mathbf{b}_k$ as labels, training dataset $\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D}$
        Obtain hyperplane $\mathbf{w}_k$ and bias t_k
    end
    $B_{ik} = \text{sgn}(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k)$ for $i = \{1 \ldots N_{trd}\}$ and $k = \{1 \ldots K\}$
end
return $\left\{\mathbf{w}_k \in \mathbb{R}^D\right\}_{k=1}^K$, $\left\{t_k \in \mathbb{R}\right\}_{k=1}^K$

non-linear variant using radial basis function kernels can be learnt on larger datasets by computing the kernel using a small subset of anchor data-points, in a similar manner to Supervised Hashing with Kernels (KSH)[4]. The projection function developed in this chapter is *agnostic* to the type of classifier used to learn the hypersurfaces. A possible future extension of the algorithm would involve scaling to a large-scale streaming data scenario, such as event detection in Twitter (Osborne et al. (2014)). In this case an online *passive aggressive* classifier (Crammer et al. (2006)) would be capable of incrementally updating the hypersurfaces in a computationally scalable fashion. I discuss avenues for possible future work in more detail in Chapter 8.

6.3 Experimental Evaluation

6.3.1 Experimental Configuration

In my experimental evaluation I adhere closely to related work so that the results in this chapter are directly comparable to previously published research. As discussed

[4] See Chapter 2, Section 2.6.4.3 for an overview of KSH.

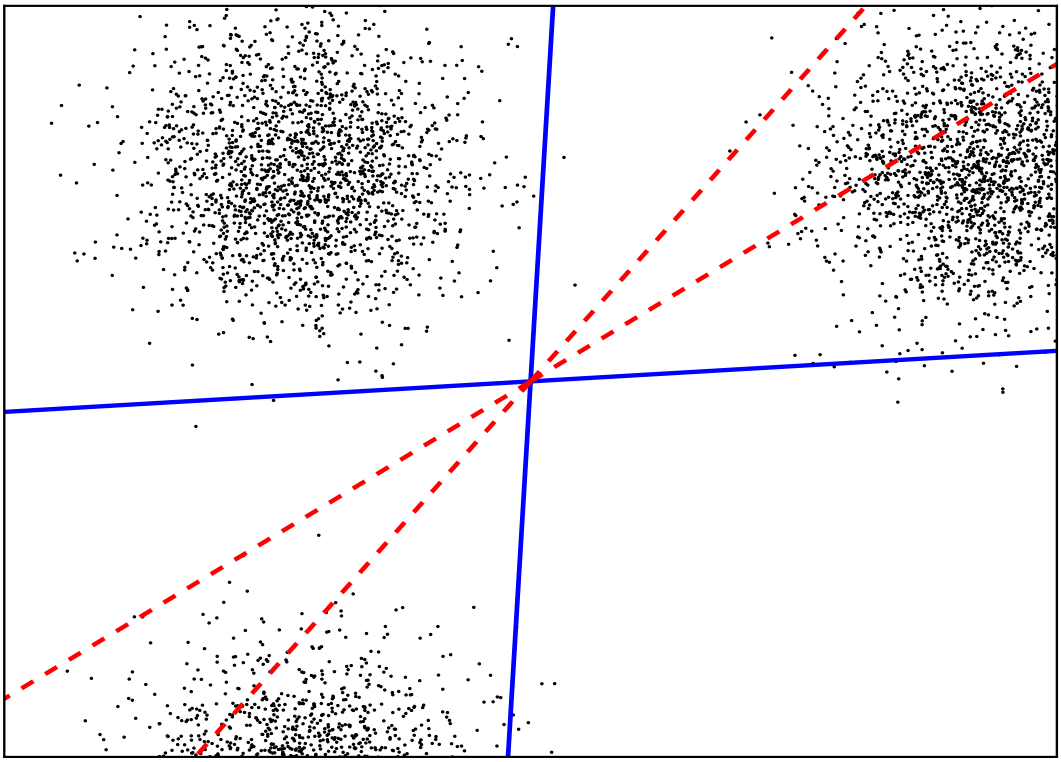


Figure 6.4: Synthetic toy example illustrating the operation of my graph regularised projection function. The toy dataset consists of 6,000 data-points clustered into three distinct clusters. The data-points in each cluster are of the same class as each other (out of three possible classes), and therefore each cluster should ideally end up in its own bucket (region). The dashed lines indicate the two hyperplane normal vectors produced by Locality Sensitive Hashing (LSH). In this case many data-points from different classes (clusters) end up in the same bucket (mAP=0.6803). GRH refines the two LSH hyperplanes to produce the hyperplane normal vectors shown with the solid lines. In this case, the data-points from the three different classes are almost all in their own bucket (mAP=0.9931).

in Chapter 3 the particular sub-field of hashing that pertains to supervised projection functions maintains a standard evaluation paradigm that differs from that of the quantisation literature to which I contributed in Chapters 4-5. The main points of differentiation are the use of human assigned *class labels* to define the groundtruth (Chapter 3, Section 3.3.2) and mean average precision (mAP) as the evaluation metric (Chapter 3, Section 3.6.4). If a query image and a retrieval image share a class in common then they are regarded as true nearest neighbours (Liu et al. (2012), Gong and Lazebnik (2011)). The Hamming ranking evaluation paradigm (Chapter 3, Section 3.4.1) is used to rank database data-points with respect to the queries for the purposes of com-

Parameter	Setting	Chapter Reference
Groundtruth Definition	Class labels, ε-NN	Chapter 3, Sections 3.3.1, 3.3.2
Evaluation Metric	mAP, PR Curve, AUPRC	Chapter 3, Section 3.6
Evaluation Paradigm	Hamming Ranking	Chapter 3, Section 3.4.1
Random Partitions	10	Chapter 3, Section 3.5
Number of Bits (K)	16-64	Chapter 2, Section 2.4

Table 6.2: Configuration of the main experimental parameters for the results presented in this chapter.

puting the retrieval metrics. I follow previous research and mostly use this evaluation strategy (class label-based groundtruth, mAP, Hamming ranking) to evaluate my graph regularised projection function on the labelled CIFAR-10 and NUS-WIDE datasets. In addition to this I also explore the effectiveness of my model on the large SIFT1M image dataset with metric-based groundtruth, that is, groundtruth generated by computing ε-NN's. The exact specification of my evaluation setup is detailed in Table 6.2.

The CIFAR-10 and NUS-WIDE datasets both come associated with manually assigned class labels that determine which images in both collections are semantically related. The NUS-WIDE dataset is one of the largest *labelled* image datasets that I am aware of in the learning to hash literature and both image datasets have been extensively used in related hashing research (Liu et al. (2012, 2011); Kulis and Grauman (2009)). For CIFAR-10 and NUS-WIDE, I carefully follow previous work in constructing my set of queries and training/database subsets shown in Tables 6.3-6.4. I randomly sample 100 images (CIFAR-10, NUS-WIDE) from *each class* to construct my testing queries $\mathbf{X}_{teq} \in \mathbb{R}^{N_{teq} \times D}$. The remaining images form the database of images to be ranked $\mathbf{X}_{ted} \in \mathbb{R}^{N_{ted} \times D}$ in accordance to the selected dataset splitting strategy (improved or literature standard) as detailed in Chapter 3, Section 3.5. I randomly sample $N_{trd} = 100/500$ images *per class* from the database ($\mathbf{X}_{db} \in \mathbb{R}^{N_{db} \times D}$) to form the training dataset ($\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D}$) which is used to learn the hash functions. The validation dataset ($\mathbf{X}_{vaq} \in \mathbb{R}^{N_{vaq} \times D}$, $\mathbf{X}_{vad} \in \mathbb{R}^{N_{vad} \times D}$) is created by sampling 100 (CIFAR-10) or 500 (NUS-WIDE) images per class from the database ($\mathbf{X}_{db} \in \mathbb{R}^{N_{db} \times D}$). For SIFT1M, which does not have class labels, I follow the experimental procedure for metric-based groundtruth outlined in Chapter 4, Section 4.3.1.

To ascertain the retrieval effectiveness of my projection function I will segment the experimental evaluation into six different hypotheses which can be directly tested

Partition	CIFAR-10	NUS-WIDE	SIFT1M
Test queries (N_{teq})	1,000	2,100	1,000
Validation queries (N_{vaq})	1,000	2,100	1,000
Validation database (N_{vad})	10,000	10,000	10,000
Training database (N_{trd})	1,000	10,500	10,000
Test database (N_{ted})	47,000	171,134	978,000

Table 6.3: Improved splitting strategy partition sizes for the experiments in this chapter. This breakdown is based on the splitting strategy introduced in Chapter 3, Section 3.5.2. There is no overlap between the data-points across partitions.

against prior-art:

- H_1: *Hyperplanes learned in a supervised manner give a higher retrieval effectiveness than randomly generated hyperplanes.*
- H_2: *Supervised hyperplanes attain a higher retrieval effectiveness than hyperplanes learnt with an unsupervised matrix factorisation.*
- H_3: *Regularising hashcodes over a data affinity graph is more effective than a Laplacian Eigenmap (LapEig) embedding for learning effective similarity preserving hashcodes.*
- H_4: *A supervised initialisation of my model with ITQ+CCA results in a higher retrieval effectiveness compared to an unsupervised initialisation through LSH.*
- H_5: *Learning non-linear hashing hypersurfaces with my model gives a higher retrieval effectiveness than learnt linear hypersurfaces (hyperplanes).*
- H_6: *Learning non-linear hashing hypersurfaces with my graph regularised projection function gives a higher retrieval effectiveness than the state-of-the-art supervised projection functions.*

The experimental results arising from testing these hypotheses are presented in Section 6.3.3. In my experiments I compare the proposed graph regularised projection function to a wide selection of strong baselines cutting across all three major branches of the hashing projection field: data-independent, unsupervised data-dependent, supervised data-dependent. The *supervised data-dependent* methods I compare to are Supervised Hashing with Kernels (KSH) (Liu et al. (2012)), Binary Reconstructive

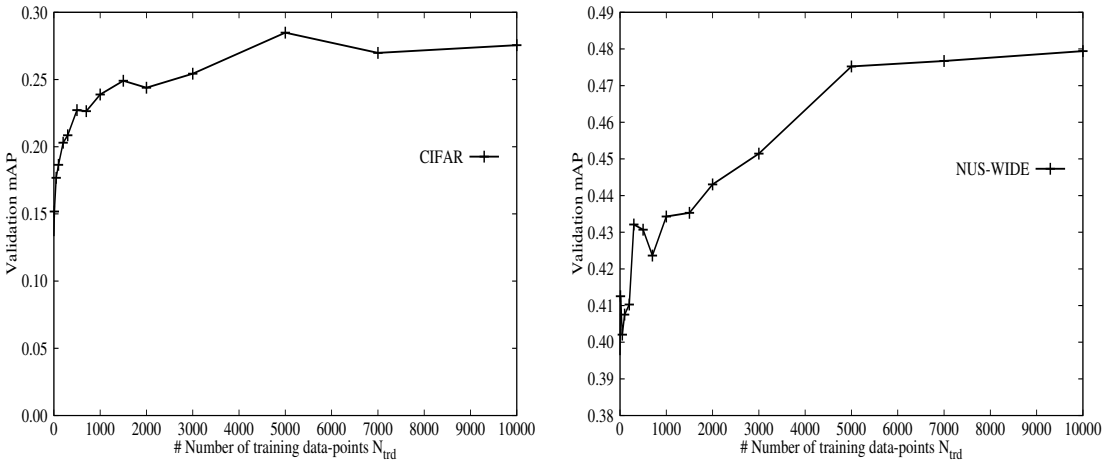
Partition	CIFAR-10	NUS-WIDE	SIFT1M
Test queries (N_{teq})	1,000	2,100	1,000
Validation queries (N_{vaq})	1,000	2,100	1,000
Validation database (N_{vad})	10,000	10,000	10,000
Training database (N_{trd})	1,000	10,500	10,000
Test database (N_{ted})	59,000	193,734	999,000

Table 6.4: Literature standard splitting strategy partition sizes for the experiments in this chapter. This breakdown is based on the splitting strategy introduced in Chapter 3, Section 3.5.1.

Embedding (BRE) (Kulis and Grauman (2009)), Self Taught Hashing (STH) (Zhang et al. (2010b)) and Iterative Quantisation (ITQ) with a supervised CCA embedding (ITQ+CCA) (Gong and Lazebnik (2011)). The *unsupervised data-dependent* projection functions include Anchor Graph Hashing (AGH) (Liu et al. (2011)), Spectral Hashing (SH) (Weiss et al. (2008)) and PCA-Hashing (PCAH) (Wang et al. (2012)). The *data-independent* methods are Locality Sensitive Hashing (LSH) (Indyk and Motwani (1998)) and Shift Invariant Kernel Hashing (SKLSH) (Kulis and Darrell (2009)). All of these projection functions were previously reviewed in detail in Chapter 2, Section 2.4 and Section 2.6. I use the standard *single bit quantisation (SBQ)* described in Chapter 2, Section 2.5.1 to binarise the projections from all projection functions in this chapter. In Chapter 7 I explore the benefit of using NPQ and VBQ introduced in Chapters 4-5 to quantise the projections arising from GRH.

6.3.2 Parameter Optimisation

The algorithm has four meta-parameters: the number of iterations $M \in \mathbb{Z}_+$, the amount of regularisation $\alpha \in [0,1]$, the flexibility of margin $C \in \mathbb{R}_+$, and the surface curvature $\gamma \in \mathbb{R}_+$, which arises for non-linear hypersurfaces based on radial-basis functions (RBFs). I optimise all meta-parameters via grid search on the held-out validation dataset ($\mathbf{X}_{vaq} \in \mathbb{R}^{N_{vaq} \times D}, \mathbf{X}_{vad} \in \mathbb{R}^{N_{vad} \times D}$) (Moran and Lavrenko (2015a)). Unless otherwise stated in the relevant experiment I tune these parameters using the following strategy in all subsequent experiments: firstly holding the SVM parameters constant at their default values ($C = 1$, $\gamma = 1.0$), I perform a grid search over $M \in \{1,2\ldots19,20\}$ and $\alpha \in \{0.1,0.2,\ldots,0.9,1.0\}$, selecting the overall configuration that leads to the highest *validation* dataset mAP. For a given α, to determine a suitable value for M, I



(a) **CIFAR-10** (b) **NUS-WIDE**

Figure 6.5: Learning curves for CIFAR-10 (Figure (a)) and NUS-WIDE (Figure (b)) as the number of training data-points N_{trd} is increased.

stop the sweep when the validation dataset mAP falls for the first time, and set M to be the number of the penultimate iteration. I then hold M and α constant at their optimised values, and perform a coarse logarithmic grid search over $\gamma \in \{0.001, 0.01, 0.1, 1, 10.0\}$ and $C \in \{0.01, 0.1, 1, 10, 100\}$. I equally weigh both classes (-1 and 1) in the SVM. To constrain computation time on NUS-WIDE I learn a *low-rank linearisation RBF SVM* with a default 300 k-means landmarks using the `budgetedSVM` library[5].

6.3.3 Experimental Results

6.3.3.1 Evaluation of Amount of Supervision (N_{trd})

In this first experiment I will examine how the maximum validation dataset mAP achieved by my graph regularised projection function varies as the number of supervisory data-points in the adjacency matrix $\mathbf{S}$ are gradually increased. To generate the learning curves I tune the parameters of my model as detailed in Section 6.3.2, while keeping the flexibility of margin of the linear SVM set to $C = 1.0$. The learning curves for both the CIFAR-10 and NUS-WIDE datasets are shown in Figure 6.5. Both curves exhibit the expected trend of increasing mAP as I give more supervision to the model. For both image collections the validation dataset mAP increases rapidly up until $N_{trd} = 5,000$ before undergoing a gentler increase as N_{trd} is further increased beyond 5,000 data-points. This experiment suggests that the majority of the perfor-

[5] `http://www.dabi.temple.edu/budgetedsvm/`

mance can be obtained with a relatively small amount of supervision totaling around or below 2-10% of the total image collection size. I arrived at a broadly similar conclusion for my multi-threshold quantisation algorithm in Chapter 4. The need for only a small amount of supervision bodes well for the scalability of my projection function. I measure the training and testing time of the model in the next experiment (Section 6.3.3.2). Even though the mAP on CIFAR-10 is clearly maximised for $N_{trd} = 5,000$, to accord with the literature (Liu et al. (2012)), I select $N_{trd} = 1,000$ for CIFAR-10, $N_{trd} = 10,500$ for NUS-WIDE and $N_{trd} = 10,000$ for SIFT1M throughout this chapter.

6.3.3.2 Evaluation of Training Time

I will now measure the training time of the supervised projection function. In a similar way to the semi-supervised quantisation algorithms introduced in Chapters 4-5 my projection model requires an *offline* training phase in which to learn the hashing hyperplanes $\left\{\mathbf{w}_k \in \mathbb{R}^D\right\}_{k=1}^{K}$. Once the hyperplanes are learnt the encoding (test) stage of the linear variant of my algorithm is as fast as any other baseline including LSH: we simply take the dot product of the data-point feature representation with the K hyperplanes and quantise the resulting projections. In this experiment I use the linear variant of my algorithm as initialised with LSH. In Table 6.5 I present the training and testing times in seconds for my algorithm (GRH) and two competitive supervised projection functions proposed in the literature (KSH and BRE). I observe that the linear variant of GRH is competitive in training and testing time to the baseline projection functions.

	Train Complexity	**Time (s)**	**Test Complexity**	**Time (s)**
GRH	$O(MN_{trd}DK + MSK)$	8.01	$O(N_{teq}DK)$	0.03
KSH	$O(KN_{trd}C + KC^3)$	74.02	$O(N_{teq}DC + N_{teq}CK)$	0.10
BRE	$O(KN_{trd}C + KN_{trd}\log N_{trd})$	227.84	$O(N_{teq}DC + N_{teq}CK)$	0.37

Table 6.5: Training and testing time on the CIFAR-10 dataset (seconds), stated as an average across all 1,000 queries over 10 independent runs. The results for the linear variant of GRH ($\text{GRH}_{lin,lsh}$) are shown. The timing results were recorded on an otherwise idle Intel 2.7GHz, 16Gb RAM machine. All models are implemented in the same software stack (Matlab). For CIFAR-10: $D = 512, C = 300, M = 4, N_{trd} = 1,000, K = 32, N_{teq} = 1,000$ (queries) and S is the number of non-zero elements in the data-point adjacency matrix $\mathbf{S}$. S is typically 10% of N_{trd}^2 for CIFAR-10 with the class-based groundtruth.

For example on CIFAR-10 at 32 bits, $\text{GRH}_{lin,lsh}$ with four iterations ($M = 4$) requires only 11% of the training time of KSH and only 3.5% of BRE while crucially exhibiting a prediction (testing) time that is approximately an order of magnitude lower than both baselines.

BRE is particularly expensive at training time requiring substantially more computation than my own model and the KSH baseline. The reason for this disparity is the coordinate descent optimisation algorithm employed by BRE to update the hashing hyperplanes during the learning procedure. As touched upon in my review of BRE in Chapter 2, Section 2.6.4.2, BRE attempts to optimise an objective function involving the sign function making it non-differentiable and therefore gradient descent inapplicable. Instead coordinate descent is used to update each element of each hyperplane individually during an iteration making the optimisation much less efficient. In my review in Chapter 2, I hinted at *three* ways in which this discrete optimisation can be relaxed: performing a coordinate descent directly in the discrete hashcode space, dropping the sign function ("spectral relaxation") or using the multi-step iterative scheme design pattern in which the hyperplanes and hashcodes are optimised individually, while keeping the other fixed. KSH implements the gradient descent method and my contribution is an example of the iterative multi-step approach. Given the timing results, I conclude that relaxing the NP-hard discrete optimisation problem leads, as might be expected, to a more computationally efficient training time. I examine the retrieval effectiveness arising from these different flavours of optimisation in Section 6.3.3.9.

6.3.3.3 Effect of the Interpolation Parameter α and Iterations M

The interpolation parameter α in Equation 6.1 determines the proportion of regularised bits from the previous iteration and the initialised bits at iteration 0 that are used to compute the regularised hashcodes at the current iteration. Adjusting α in the range $[0, 1]$ determines how much weight we place on smoothing the bits using the image adjacency graph compared to maintaining a consistency with the initial hashcodes. In some sense this parameter is reminiscent of the interpolation parameter used in the multiple threshold quantisation model introduced in Chapter 4 in which an interpolation was made between a supervised and unsupervised signal. In that chapter I found the supervised signal to be much more important in general than the unsupervised signal for the purposes of effective hashcode generation. In this section I conduct a similar investigation for the α parameter in the context of my graph regularised projection function. I also jointly examine the effect of the number of iterations M since

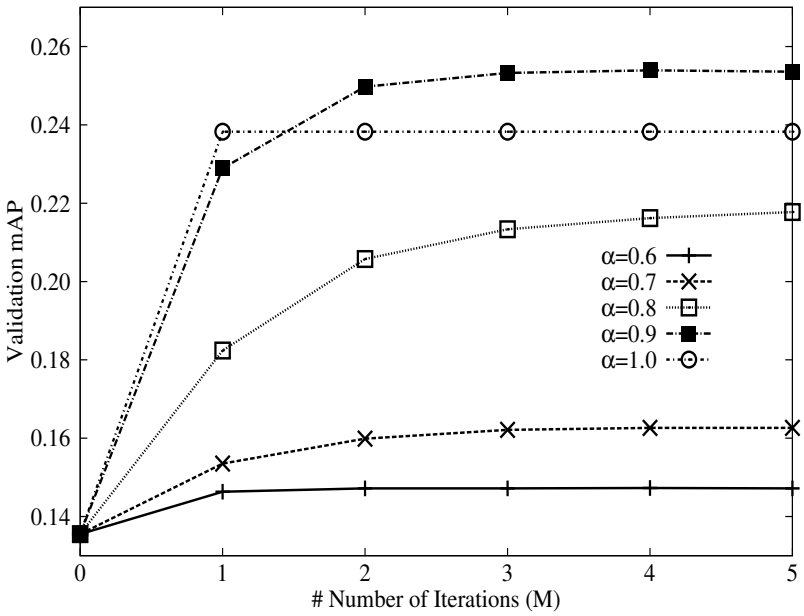
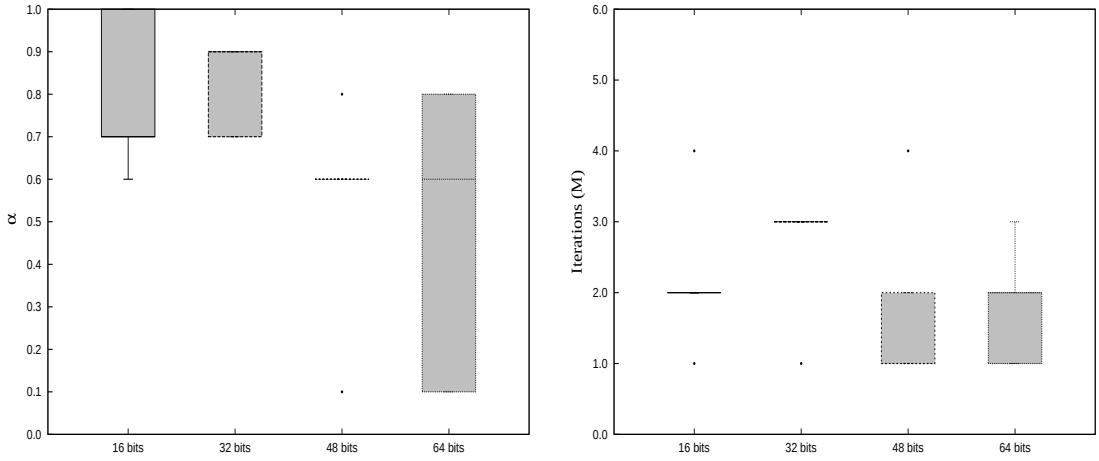


Figure 6.6: CIFAR-10 validation mAP at 32 bits versus iterations (M) and the setting of the interpolation parameter α. GRH is parameterised with a linear kernel and an LSH initialisation. The setting of α is more important to final retrieval effectiveness than the value of M.

the optimal setting of both parameters is likely to be tied given their existence within the central optimisation loop of my algorithm. To isolate the effect of α and M only, I keep the linear SVM flexibility of margin set to $C = 1.0$ throughout this experiment.

In Figure 6.6 I present a set of results that illustrate how the retrieval effectiveness on the CIFAR-10 validation dataset is influenced by the setting of the α and M parameters. To create the validation dataset I reserved a randomly selected proportion of the CIFAR-10 dataset to form the validation dataset queries ($\mathbf{X}_{vaq} \in \mathbb{R}^{N_{vaq} \times D}$, $N_{vaq} = 1,000$) and validation dataset database ($\mathbf{X}_{vad} \in \mathbb{R}^{N_{vad} \times D}$, $N_{vad} = 10,000$) against which those queries were run. I observe that the retrieval effectiveness appears to depend largely on the setting of α, and less so on the number of iterations M. The highest validation mAP is achieved with $\alpha = 0.9$ in the case of this particular random validation dataset split. This result agrees with expectations in that it is much more important to smooth the bits with the adjacency graph than it is to maintain a consistency with the initialised hashcodes $\mathbf{B}_0$.

Given that the optimal α may vary depending on the random split of the dataset, on the hashcode length and on the manner of initialisation (LSH or ITQ+CCA), I conducted further experiments to verify the preferred setting of the parameter on the



(a) **Interpolation (α)** (b) **Iterations (M)**

Figure 6.7: Box plots displaying the variation in the α value (Figure (a)) and the number of iterations (M) (Figure (b)) across different hashcode lengths. Results are for the CIFAR-10 dataset with an ITQ+CCA embedding.

CIFAR-10 dataset. To do so I swept $\alpha \in [0.1, 0.2, 0.3, \ldots, 0.9, 1.0]$ over five random validation dataset splits ($\mathbf{X}_{vaq}$, $\mathbf{X}_{vad}$) and found the value of α that yielded the maximum validation dataset mAP for each split. This procedure was repeated for four different hashcode lengths $K \in [16, 32, 48, 64]$ and for two different initialisation methods (LSH and ITQ+CCA). For an LSH initialisation of the hashcodes, I found the minimum and maximum value of α to be 0.9 and 1.0 respectively, with a modal value of $\alpha = 1.0$. The situation is rather different when I initialise my model with a supervised ITQ+CCA embedding (Figure 6.7a). In this case much lower values of α are frequently optimal (modal value of $\alpha = 0.6$) thereby placing more weight on the initialised hashcodes during the optimisation procedure. This finding is quite intuitive given that the supervised embedding generally provides a much better initialisation point for the optimisation than hashcodes generated from a random spatial partitioning. Nevertheless, in both cases (LSH and ITQ+CCA), I find that the median α value is always above 0.5 thereby giving a greater proportion of the weight to the supervisory signal.

I now explore the preferred setting of the number of iterations (M) in more detail. In a similar manner to my exploration of the α parameter I form five different random validation dataset splits ($\mathbf{X}_{vaq}$, $\mathbf{X}_{vad}$). For each of those five validation datasets I find the value of iteration number at which the validation mAP first falls. The preferred value of M is then set to be the number of the previous iteration. I repeat this pro-

cedure for four different hashcode lengths (16, 32, 48 and 64 bits). In general, I find that the best setting of M for CIFAR-10 is always found to be between 1-5 iterations with a median of 2 iterations for all considered hashcode lengths and both methods of initialisation (LSH or ITQ+CCA). In Figure 6.7b I illustrate this by showing the box plot depicting the variation in M for an ITQ+CCA initialisation. The fact that the optimisation reaches the highest validation mAP within a low number of iterations is encouraging from an efficiency standpoint and is in fact one reason the training time of my iterative model is a fraction of that exhibited by comparable baselines (Section 6.3.3.2).

6.3.3.4 Experiment I: Learnt Hyperplanes versus Random Hyperplanes

In this experiment I compare the graph regularised projection function against two well-known randomised projection functions that both draw the hashing hyperplanes randomly within the input feature space. My baselines are Locality Sensitive Hashing (LSH) and Shift Invariant Kernel Hashing (SKLSH) both of which were reviewed in Chapter 2, Section 2.4.1 and Section 2.6.2.1, respectively. SKLSH and the inner product similarity version of LSH both draw K hyperplanes randomly from a zero mean unit variance multidimensional Gaussian distribution. The distribution of the input data is therefore not considered during the spatial partitioning of the input feature space, with both algorithms depending on an asymptotic guarantee that as the number of hashcode bits is increased the desired similarity will be preserved in the generated hashcodes. In this section I contest that this random partitioning of the input feature space does not lend itself well to the generation of compact and discriminative hashcodes for image retrieval.

CIFAR-10 (Experiment I)			
	16 bits	**32 bits**	**64 bits**
LSH	0.1290 (0.1327)	0.1394 (0.1403)	0.1525 (0.1531)
SKLSH	0.1145 (0.1174)	0.1199 (0.1182)	0.1269 (0.1270)
GRH$_{lin,lsh}$	**0.2149** (0.2137)▲▲	**0.2443** (0.2427)▲▲	**0.2460** (0.2445)▲▲

Table 6.6: mAP on the CIFAR-10 image dataset for GRH and the LSH and SKLSH baselines. lin: linear kernel, lsh: LSH initialisation. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over LSH. The improved splitting strategy result is shown in brackets.

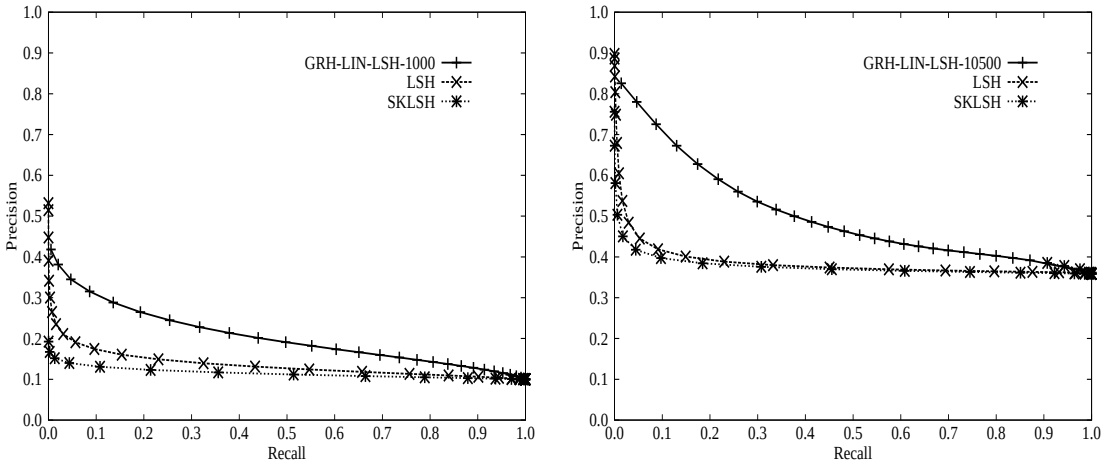
NUS-WIDE (Experiment I)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
LSH	0.3837	0.3883	0.3863	0.3879
SKLSH	0.3724	0.3755	0.3734	0.3827
GRH$_{lin,lsh}$	**0.4928▲▲**	**0.4971▲▲**	**0.5023▲▲**	**0.5065▲▲**

Table 6.7: mAP for the NUS-WIDE dataset. lin: linear kernel, lsh: LSH initialisation. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over LSH.

SIFT1M (Experiment I)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
LSH	0.0273	0.1051	0.2023	0.2524
GRH$_{lin,lsh}$	**0.0388▲**	**0.1569▲▲**	**0.2980▲▲**	**0.3860▲▲**

Table 6.8: AUPRC for the SIFT1M dataset. lin: linear kernel, lsh: LSH initialisation. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over LSH. ▲/▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.05$) over LSH.

In Tables 6.6-6.8 and Figure 6.8 I present the performance achieved by my own method (GRH$_{lin,lsh}$) and the two considered baselines (LSH, SKLSH) on the CIFAR-10, NUS-WIDE and SIFT1M image collections. The parameters of my model (M,α,C) are optimised in accordance with the procedure outlined in Section 6.3.2. On all three datasets I observe that my graph regularised projection function decisively outperforms both baselines across all considered hashcode lengths yielding the highest overall performance in each case. For example, at a hashcode length of 32 bits on CIFAR-10, my algorithm for learning the hashing hyperplanes achieves a considerable 75% increase in mAP compared to LSH. The precision recall curve for GRH dominates those of LSH and SKLSH at all recall levels (Figure 6.8a). A similar pattern is found on the larger NUS-WIDE dataset, where at 32 bits GRH both attains a 28% increase in mAP (Table 6.7) and dominates at all levels of recall (Figure 6.8b), and the SIFT-1M dataset with metric-based supervision (Table 6.8). I confirm my first hypothesis (H_1) that learnt hyperplanes can achieve a higher retrieval effectiveness with compact hashcodes than randomly placed hyperplanes. I note that this experimental result accords with the wealth of evidence presented in the supervised projection function literature



(a) **CIFAR-10** (b) **NUS-WIDE**

Figure 6.8: Precision recall curves for the CIFAR-10 dataset (Figure (a)) and the NUS-WIDE dataset ((b)) for a hashcode length of 32 bits. Results are for Experiment I.

as to the higher effectiveness of learnt hyperplanes versus random hyperplanes (Liu et al. (2012), Gong and Lazebnik (2011)). A selection of qualitative results comparing the top ten ranked images for LSH and $\text{GRH}_{lin,lsh}$ are presented in Tables 6.9-6.12.

Finally, it is apparent in Table 6.6 that the mAP figures resulting from the literature standard and improved splitting strategies described in Chapter 3, Section 3.5 are effectively similar and importantly do not change the ranking of the algorithms (from best to worst). This finding corroborates my earlier results in Chapters 4-5 in which I also found that there was no significant difference between the results obtained from both methods of splitting the dataset into testing and database partitions. In this chapter I will again therefore only report the retrieval results arising from the literature standard dataset splitting strategy.

6.3.3.5 Experiment II: Supervised versus Data-Dependent (Unsupervised) Hyperplane Learning

In the previous experiment I found that hashcodes generated from hyperplanes that were learnt based on class labels were more effective than those derived from randomly generated hyperplanes. In this experiment I will compare hyperplanes learnt with a supervisory signal to hyperplanes that are learnt using an unsupervised matrix factorisation such as a Laplacian Eigenmap (LapEig) or Principal Components Analysis (PCA). In my review in Chapter 2, Section 2.6.3 I described four projection functions for hashing that all leveraged a matrix factorisation to learn hyperplanes: PCAH,

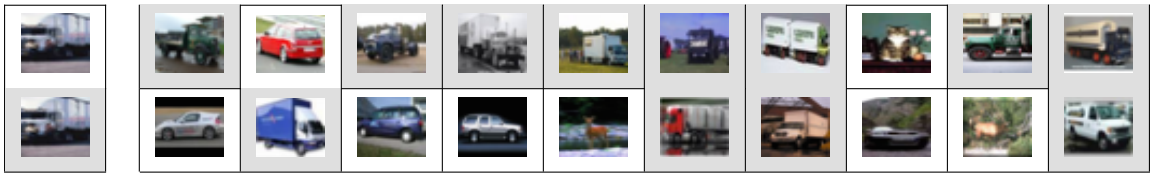


Table 6.9: **Left-most column:** *Truck* query image. **Top row:** GRH top 10 retrieved images, precision: 0.8. **Bottom row:** LSH top 10 retrieved images, precision: 0.4. Shaded cells indicated true positives.

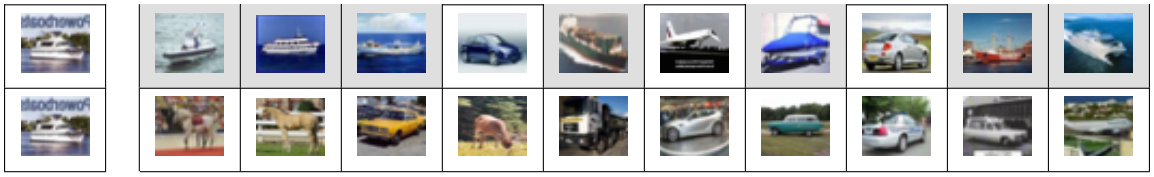


Table 6.10: **Left-most column:** *Boat* query image. **Top row:** GRH top 10 retrieved images, precision: 0.5. **Bottom row:** LSH top 10 retrieved images, precision: 0.0.

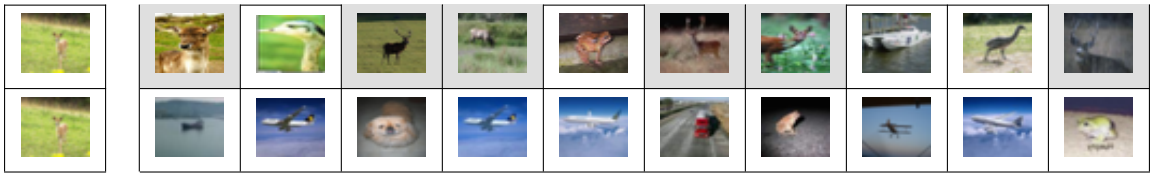


Table 6.11: **Left-most column:** *Deer* query image. **Top row:** GRH top 10 retrieved images, precision: 0.5. **Bottom row:** LSH top 10 retrieved images, precision: 0.0.

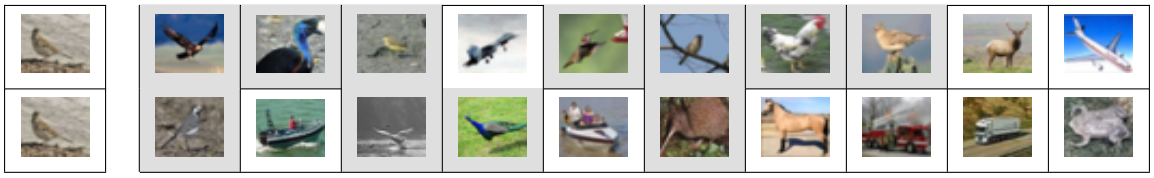


Table 6.12: **Left-most column:** *Bird* query image. **Top row:** GRH top 10 retrieved images, precision: 0.7. **Bottom row:** LSH top 10 retrieved images, precision: 0.4.

CIFAR-10 (Experiment II)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
PCAH	0.1322	0.1291	0.1256	0.1234
SH	0.1306	0.1296	0.1346	0.1314
ITQ	0.1631	0.1699	0.1732	0.1743
AGH	0.1616	0.1577	0.1599	0.1588
GRH$_{lin,lsh}$	**0.2149▲▲**	**0.2443▲▲**	**0.2433▲▲**	**0.2460▲▲**

Table 6.13: mAP on the CIFAR-10 image dataset. lin: linear kernel, lsh: LSH initialisation. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over ITQ.

SH, ITQ and AGH. Three of these methods (ITQ, SH, PCAH) leverage PCA to learn a set of hyperplanes that capture the maximum variance in the input feature space, while two of those (ITQ, SH) then further attempt to balance the variance across hyperplanes by rotating the data (ITQ) or assigning more bits to higher variance hyperplanes (SH). The importance of exploiting the more reliable higher variance hyperplanes was discussed in detail in Chapter 5 in the context of my own variable threshold quantisation algorithm. In contrast, AGH instead leverages the Laplacian Eigenmap dimensionality reduction method to learn data-dependent hyperplanes. In this experiment I will discover how these unsupervised dimensionality reduction methods compare against hashing hyperplanes that are learnt based on a supervisory signal. I hypothesise that supervision is critical to retrieval effectiveness and that relying on an unsupervised signal such as variance to learn discriminative hyperplanes is sub-optimal. Note as for all projection functions in this chapter I use single bit quantisation (SBQ) to binarise the projections.

The experimental results are presented in Table 6.13 for the CIFAR-10 dataset and in Table 6.14 for the NUS-WIDE dataset. Across both datasets and all considered hashcode lengths I see that hashcodes generated from hyperplanes learnt with a supervisory signal (GRH$_{lin,lsh}$) significantly (Wilcoxon signed rank test, $p < 0.01$) outperform hashcodes learnt from a matrix factorisation (PCAH, SH, ITQ, AGH). I confirm hypothesis H_2 and conclude from these experimental results that it is critical to integrate a degree of supervision into a hashing projection function so as to maximise retrieval effectiveness.

NUS-WIDE (Experiment II)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
PCAH	0.3890	0.3863	0.3829	0.3804
SH	0.3734	0.3751	0.3760	0.3751
ITQ	0.4112	0.4136	0.4148	0.4164
AGH	0.3820	0.3809	0.3782	0.3767
GRH$_{lin,lsh}$	**0.4928▲▲**	**0.4971▲▲**	**0.5023▲▲**	**0.5065▲▲**

Table 6.14: mAP on the NUS-WIDE image dataset. lin: linear kernel, lsh: LSH initialisation. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon signed rank test, $p < 0.01$) over ITQ.

6.3.3.6 Experiment III: Graph Regularisation versus Laplacian Eigenmap (LapEig) Embedding

In this experiment, I am interested in isolating the effect of the graph regularisation component of my model. Recall from Section 6.2.3 that in Step A of the algorithm the hashcodes at the current iteration are smoothed (regularised) over the image adjacency graph encoded in the matrix $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$. I hypothesise that this graph regularisation component is the critical factor in the effectiveness of my projection function. To investigate this hypothesis I compare directly to the Self Taught Hashing (STH) model of Zhang et al. (2010b). In Chapter 2, Section 2.6.4.4, I reviewed STH in detail. To summarise, STH first projects the training data-points into a lower K dimensional space using the Laplacian Eigenmap (LapEig) dimensionality reduction technique. LapEig takes as input an adjacency graph encoding the pairwise relationship between the training data-points and attempts to construct an embedding space in which data-points close by in $\mathbf{S}$ are close by in the embedding space. The STH model then binarises the resulting Laplacian eigenvectors to form N_{trd} K-bit hashcodes, one hashcode for each of the N_{trd} training data-points. The bits of the hashcodes are subsequently used as the labels to learn K binary SVM classifiers, the weight vectors of which form the hashing hyperplanes. The STH baseline is therefore the most closely related projection function to my own model. The main difference between STH and my own model lies in the use of LapEig and graph regularisation to maintain the neighbourhood structure, respectively. By directly comparing both models on the task of image retrieval I can therefore measure which method of maintaining the neighbourhood structure between the data-points is more effective, namely graph regularisation

CIFAR-10 (Experiment III)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
STH$_{lin}$	0.1713	0.1848	0.1806	0.1891
GRH$_{lin,lsh}$	**0.2149▲▲**	**0.2443▲▲**	**0.2433▲▲**	**0.2460▲▲**

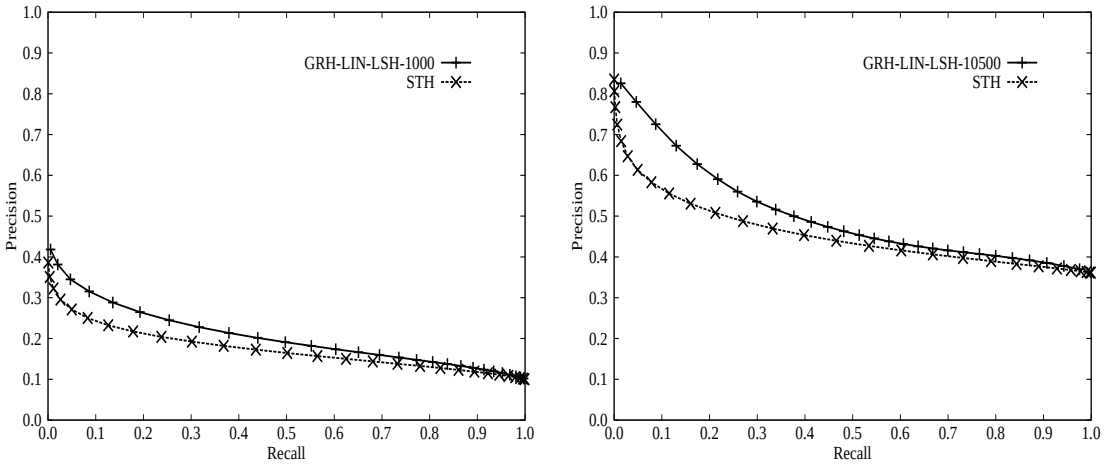
Table 6.15: mAP on the CIFAR-10 image dataset. lin: linear kernel, lsh: LSH initialisation. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over STH.

or LapEig.

To ensure a fair comparison between both models I use the same data-point adjacency graph $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}, N_{trd} = 1,000$ for both STH and my own model. I also tune the linear SVM parameters of STH in the same way I tune the linear SVM parameters of my own model (Section 6.3.2). The retrieval results from this experiment are shown in Tables 6.15-6.16 and in Figure 6.9. It is clear that my own model incorporating the graph regularisation component achieves consistently better retrieval effectiveness across both image datasets. Regularising hashcodes over an adjacency graph is therefore much more effective at learning hyperplanes for the purposes of hashing-based ANN search than is applying a LapEig dimensionality reduction. Furthermore, as STH also uses SVMs trained with hashcodes as targets, this result demonstrates that the gain realised by my model is not simply due to the use of SVMs for hyperplane learning. These findings, coupled with that in the previous experiment (Section 6.3.3.5), effectively negates the need for using a matrix factorisation to learn data-dependent hashcodes, a considerable computational bottleneck in the current breed of data-dependent hashing models. To the best of my knowledge this is the first time that a result of this nature has been reported in the learning to hash literature.

NUS-WIDE (Experiment III)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
STH$_{lin}$	0.4470	0.4593	0.4656	0.4629
GRH$_{lin,lsh}$	**0.4928▲▲**	**0.4971▲▲**	**0.5023▲▲**	**0.5065▲▲**

Table 6.16: mAP on the NUS-WIDE image dataset. lin: linear kernel, lsh: LSH initialisation. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over STH.



(a) **CIFAR-10** (b) **NUS-WIDE**

Figure 6.9: Precision recall curves for the CIFAR-10 dataset (Figure (a)) and the NUS-WIDE dataset (Figure (b)) for a hashcode length of 32 bits. Results are for Experiment III.

6.3.3.7 Experiment IV: Supervised versus Unsupervised Initialisation

The graph regularised projection function requires an initial set of hashcodes $\mathbf{B}_0 \in \{0,1\}^{N_{trd} \times N_{trd}}$ to be used as an initialisation point for the optimisation. Any existing hash function, data-independent or dependent, can be used in this context. In this experiment I seek to understand the effect on retrieval effectiveness arising from the method of hashcode initialisation. I hypothesise that a supervised initialisation will yield a higher retrieval effectiveness than an unsupervised initialisation due to the better starting point in weight space. To this end, I select LSH as a representative data-independent projection function and ITQ+CCA as a data-dependent projection function for hashcode initialisation. Note the number of GRH iterations M is optimised separately for both modes of initialisation on the held-out validation dataset, and therefore may not be identical. The retrieval results arising from these two modes of initialisation are shown in Table 6.17 for CIFAR-10 and Table 6.18 for NUS-WIDE. For CIFAR-10 the supervised initialisation ($\text{GRH}_{lin,cca}$) yields a significantly higher retrieval effectiveness (Wilcoxon signed rank test, $p < 0.01$) than the unsupervised initialisation ($\text{GRH}_{lin,lsh}$) for 16, 48 and 64 bits, but not for 32 bits (Table 6.17). On this particular dataset the manner of initialisation therefore gives a significant, albeit relatively small boost to the final retrieval performance of my algorithm. I can therefore confirm hypothesis H_4 and that supervised initialisation can have a positive effect

CIFAR-10 (Experiment IV)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
$\mathbf{GRH}_{lin,lsh}$	0.2149	0.2443	0.2433	0.2460
$\mathbf{GRH}_{lin,cca}$	**0.2403▲▲**	**0.2508**	**0.2596▲▲**	**0.2643▲▲**

Table 6.17: mAP on the CIFAR-10 image dataset. lin: linear kernel, lsh: LSH initialisation, cca: ITQ+CCA initialisation. ▲▲ denotes statistical significance (Wilcoxon signed rank, $p < 0.01$) versus $\text{GRH}_{lin,lsh}$.

NUS-WIDE (Experiment IV)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
$\mathbf{GRH}_{lin,lsh}$	0.4928	0.4971	0.5023	**0.5065**
$\mathbf{GRH}_{lin,cca}$	**0.4969**	**0.4985**	**0.5027**	0.5000

Table 6.18: mAP on the NUS-WIDE image dataset. lin: linear kernel, lsh: LSH initialisation, cca: ITQ+CCA initialisation.

on retrieval effectiveness. On the NUS-WIDE dataset, the manner of initialisation has no significant effect as observed in Table 6.18. It is comforting that the majority of the observed performance on this dataset can be achieved with a random LSH initialisation, thereby allowing one to avoid an initialisation based on an expensive matrix factorisation.

6.3.3.8 Experiment V: Non-Linear Hypersurfaces versus Linear Hypersurfaces

The semantic relationship between images is likely to be complex and highly non-linear in the input feature space. Linear hypersurfaces due to their restriction to forming planes in the feature space may not provide as good a partitioning of the space compared to non-linear hypersurfaces that have a greater degree of freedom. My model maintains the flexibility of using either linear or non-linear hypersurfaces to partition the input feature space. In this experiment I hypothesise, due to the enhanced flexibility of the induced decision boundaries, that non-linear hypersurfaces will lead to a higher retrieval effectiveness than hyperplanes (linear hypersurfaces). Recall from Chapter 2 that linear hash functions employing hyperplanes as a means of spatial partitioning are by far the most common hash functions in the learning to hash literature. I contend that the greater flexibility exhibited by non-linear hypersurfaces is more likely to lead to a better spatial partitioning of the data in which more related data-points fall within

the same regions. Given that data-points within the same enclosed regions formed by the non-linear hypersurfaces will have the same hashcodes, I therefore expect a higher retrieval effectiveness to arise from a more refined spatial partitioning. I perform a set of experiments in this section to test this claim.

To learn non-linear hypersurfaces I simply replace the linear kernel in the dual form of the SVM with a non-linear radial basis function (RBF) kernel[6]. The standard dual formulation of the SVM is specified in Equation 6.4 (Bishop (2006))

$$\begin{aligned} \arg\min_{\alpha} \quad & \sum_{i=1}^{N_{trd}} \alpha_i - \frac{1}{2} \sum_{i,j}^{N_{trd}} B_{ik} B_{jk} \alpha_i \alpha_j \mathbf{x}_i^\intercal \mathbf{x}_j \\ \text{subject to } & 0 \leq \alpha_i \leq C \qquad\qquad i \in \{1 \ldots N_{trd}\} \\ & \sum_{i=1}^{N_{trd}} \alpha_i B_{ik} = 0 \end{aligned} \tag{6.4}$$

The inner product $(\mathbf{x}_i^\intercal \mathbf{x}_j)$ an be replaced by a non-linear kernel such as the radial basis function (RBF) kernel given in Equation 6.5

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2) \tag{6.5}$$

where $\gamma \in \mathbb{R}_+$ is the kernel bandwidth parameter controlling the width or extent of the kernel within the feature space. The ease with which my projection function can be transformed from a linear to a non-linear model is a useful advantage compared to previous work. The power of the RBF kernel comes from the use of the "kernel trick" (Bishop (2006)) to implicitly map the input data to a higher dimensional feature space $\mathbb{R}^{D'}$ $(D' \gg D)$ in which the data is more likely to be linearly separable by linear decision boundaries. These hyperplanes in the higher dimension space form complex non-linear decision boundaries (hypersurfaces) in the original feature space $\mathbb{R}^D$. Figure 6.10 illustrates the differences in the spatial partitionings possible through linear and non-linear hypersurfaces on the CIFAR-10 dataset. The non-linear hash function can then be specified as in Equation 6.6.

$$h_k(\mathbf{x}_i) = sgn(\sum_{j=1}^{N_{trd}} \alpha_i B_{ik} \kappa(\mathbf{x}_i, \mathbf{x}_j)) \tag{6.6}$$

This is by no means the first time a kernelised hash function has been reported in the literature. For example, Liu et al. (2012) introduced a non-linear hash function that exhibited exemplary performance on image retrieval tasks, while Zhang et al.

[6]Other SVM kernels such as the polynomial kernel can easily be incorporated in my model in a similar fashion. I leave investigation of these additional kernels to future work.

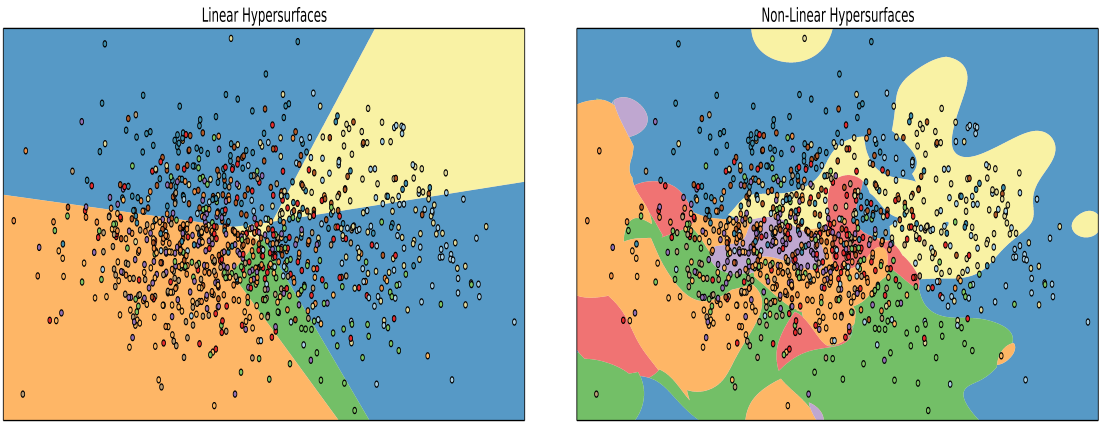


Figure 6.10: The differences between the decision boundaries induced by linear hypersurface (left) and non-linear hypersurfaces (right) on a 2D PCA projection of the CIFAR-10 dataset with 10 classes. Data-points and their classes are indicated by the coloured dots. Regions are coloured according to the predicted class for that region, and $\gamma = 0.07$ for the RBF kernel. This illustration is simply a 2D slice of the original 512 dimensional feature space. The data-points therefore may not be classed into the region within which they appear to lie within the 2D space as they are unlikely to lie on the plane spanned by the two PCA principal components. This example is inspired by a similar example presented at: `http://scikit-learn.org/stable/auto_examples/plot_kernel_approximation.html` (URL accessed on 24/3/16).

(2010a) made a non-linear extension to their Self Taught Hashing model first described in Zhang et al. (2010b). I compare my non-linear kernelised hash function to this model (KSH) in Section 6.3.3.9, and for the moment I focus on determining whether or not more complex decision boundaries translate to enhanced retrieval effectiveness for hashing-based ANN search.

I note here a significant downside to the use of non-linear hashing hypersurfaces: the substantial increase in the time required to encode novel data-points. For the linear hypersurfaces we need only perform K dot products, one for each of the K hyperplanes giving an $O(KD)$ time complexity. For the non-linear hypersurfaces this increases to $O(N_{trd}D)$, raising a serious question regarding the scalability of the non-linear model to large-scale datasets. Recent research within the machine learning literature has proposed new models for mitigating the computational complexity of learning non-linear hypersurfaces on large datasets. For example, Zhang et al. (2012) introduce the Low-rank Linearisation SVM (LLSVM) that only computes distances between the novel data-point and C k-means centers rather than the full N_{trd} training data-points giving

CIFAR-10 (Experiment V)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
GRH$_{lin,cca}$	0.2403	0.2508	0.2596	0.2643
GRH$_{rbf,300,cca}$	0.2718▲▲	0.2895▲▲	0.3026▲▲	0.3130▲▲
GRH$_{rbf,full,cca}$	**0.2991▲▲**	**0.3122▲▲**	**0.3252▲▲**	**0.3350▲▲**

Table 6.19: mAP on the CIFAR-10 image dataset. lin: linear kernel, lsh: LSH initialisation, 300: landmark (approximate) RBF with 300 centers, $full$: standard RBF. ▲▲ denotes statistical significance (Wilcoxon signed rank, $p < 0.01$) versus GRH$_{lin,cca}$.

.

an encoding (prediction) time complexity of $O(CD)$. This approach is reminiscent of Anchor Graph Hashing (AGH) and Supervised Hashing with Kernels (KSH), both of which reduce the number of pairwise comparisons by condensing the training dataset into C k-means centroids. See Chapter 2, Section 2.6.3.4 and Section 2.6.4.3 for further detailed information on AGH and KSH. In this section, unless otherwise indicated, I use the `budgetedSVM` implementation of the LLSVM with $C = 300$ k-means centers (landmarks) to learn non-linear hashing hypersurfaces on the CIFAR and NUS-WIDE datasets. This model variant is denoted as GRH$_{rbf,300,cca}$. Furthermore, on the smaller CIFAR-10 dataset I also investigate the retrieval effectiveness that is possible if I use the full $N_{trd} \times N_{trd}$ kernel matrix. This model is indicated as GRH$_{rbf,full,cca}$ in Table 6.19.

In Tables 6.19-6.20 I present the experimental results comparing the linear and non-linear versions of my model. In each case I initialise the hashcodes with ITQ+CCA and set the parameters of both models on the validation dataset ($\mathbf{X}_{vaq}$, $\mathbf{X}_{vad}$) using the search strategy detailed in Section 6.3.2. I observe on the CIFAR-10 dataset that the non-linear models (GRH$_{rbf,300,cca}$, GRH$_{rbf,full,cca}$) significantly (Wilcoxon signed rank test, $p < 0.01$) outperform the linear variant (GRH$_{lin,cca}$) across hashcode lengths of 16-64 bits. For example, at 32 bits GRH$_{rbf,full,cca}$ attains a 24% relative increase in mAP on CIFAR-10 versus GRH$_{lin,cca}$. No significant difference, however, is found between the linear and non-linear variant on the NUS-WIDE dataset. The result on the CIFAR-10 dataset supports my hypothesis (H_5) that the greater flexibility of the non-linear hypersurfaces can result in more discriminative hashcodes for the purposes of hashing-based ANN search. To conclude, even though I found that the linear variant of my model has significantly lower retrieval effectiveness than the non-linear variant,

NUS-WIDE (Experiment V)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
$\mathbf{GRH}_{lin,cca}$	0.4969	0.4985	0.5027	0.5000
$\mathbf{GRH}_{rbf,full,cca}$	**0.4996**	**0.5144**	**0.5217**	**0.5269**

Table 6.20: mAP on the NUS-WIDE image dataset. lin: linear kernel, cca: ITQ+CCA initialisation.

I am encouraged that the mAP achieved by the *linear version* is nevertheless competitive, achieving roughly 80% of the mAP of the non-linear variant on CIFAR-10 and and indistinguishable performance on NUS-WIDE. This suggests we can still maintain the attractive scalability of the linear learner while sacrificing a modicum of accuracy. If spare CPU cycles are available and the highest retrieval accuracy is important, my model can be used with non-linear hypersurfaces in a very straightforward manner to further enhance effectiveness.

6.3.3.9 Experiment VI: Comparing to state-of-the-art Supervised Projection Functions

I have so far established that hyperplanes learnt using a degree of supervision can significantly outperform data-dependent (unsupervised) projection functions that rely on a matrix factorisation (AGH, PCA, ITQ) and projection functions that randomly partition the input feature space (LSH, SKLSH). In this final experiment I now compare my model to state-of-the-art fully supervised projection functions: Supervised Hashing with Kernels (KSH), Binary Reconstructive Embedding (BRE), ITQ+CCA and Self Taught Hashing (STH) all learnt using an identical level of supervision as my own model. These projection functions were reviewed in Chapter 2, Section 2.6.4. KSH, BRE and STH can be further configured to learn non-linear hypersurfaces in a similar manner to my own model and so the learning capacity of the baselines are now on a similar footing. To ensure a fair comparison between the kernelised models (BRE, KSH and GRH_{rbf}) I configure all three models to use $C = 300$ k-means anchor points to compute the kernel. In this section I will therefore gain an insight into the effect of the style of optimisation framework on the resulting retrieval effectiveness.

Recall from Chapter 2, Section 2.6.4 that there are effectively *three* options when learning the hashing hypersurfaces: retain the sign function in the optimisation framework and attempt to learn directly within the discrete hashcode space, drop the sign

CIFAR-10 (Experiment VI)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
ITQ+CCA	0.2015	0.2130	0.2208	0.2237
STH$_{rbf}$	0.2352	0.2072	0.2118	0.2000
BRE$_{rbf}$	0.1659	0.1784	0.1904	0.1923
KSH$_{rbf}$	0.2496	0.2785	0.2849	0.2905
GRH$_{rbf,300,cca}$	0.2718▲▲	0.2895▲	0.3026▲▲	0.3130▲▲
GRH$_{rbf,full,cca}$	**0.2991▲▲**	**0.3122▲▲**	**0.3252▲▲**	**0.3350▲▲**

Table 6.21: mAP on the CIFAR-10 image dataset. rbf: radial basis function kernel, cca: ITQ+CCA initialisation. ▲▲ denotes statistical significance (Wilcoxon signed rank, $p < 0.01$) versus KSH$_{rbf}$. ▲ denotes statistical significance (Wilcoxon signed rank, $p < 0.05$) versus KSH$_{rbf}$.

function and optimise a continuous surrogate or employ an iterative multi-step scheme. BRE employs the first strategy relying on coordinate descent to update the hyperplanes, while KSH uses the second optimisation strategy and performs gradient descent using a continuous approximation to the hashcodes. In both cases BRE and KSH adjust the hypersurfaces in an attempt to minimise the difference between the labels and the hashcode distances. My model (and ITQ+CCA) is an example of the third option in which the optimisation of the hascodes (Step A: regularisation) is separated from the updating of the hypersurfaces (Step B: partitioning). This multi-step strategy allows us to neatly avoid directly optimising an NP-hard objective involving the non-differentiable sign function.

In this experiment, I compare my model to the state-of-the-art supervised hashing models on the task of image retrieval. To do so, I give all models the same amount of supervision ($N_{trd} = 1,000$ for CIFAR-10, $N_{trd} = 10,500$ for NUS-WIDE) and parameterise each to learn non-linear hypersurfaces (for ITQ+CCA the latter is not possible). The retrieval results arising from this experiment are shown in Table 6.21 for CIFAR-10 and Table 6.22 for NUS-WIDE. I can make three claims from these results: firstly a multi-step iterative scheme (GRH) can be much more effective (Wilcoxon signed rank test, $p < 0.01$) than a coordinate descent procedure (BRE) across both datasets. Secondly it is encouraging that my proposed model significantly outperforms (Wilcoxon signed rank test, $p < 0.01, 0.05$) the state-of-the-art KSH model on the CIFAR-10 dataset across all hashcode lengths. The GRH$_{rbf,300,cca}$ variant is parameterised with

NUS-WIDE (Experiment VI)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
ITQ+CCA	0.4268	0.4186	0.4161	0.4101
STH$_{rbf}$	0.4320	0.4499	0.4322	0.4305
BRE$_{rbf}$	0.4476	0.4650	0.4736	0.4776
KSH$_{rbf}$	0.4849	0.4912	0.4976	0.5018
GRH$_{lin,cca}$	0.4969	0.4985	0.5027	0.5000
GRH$_{rbf,full,cca}$	**0.4996**	**0.5144**	**0.5217**	**0.5269**

Table 6.22: mAP on the NUS-WIDE image dataset. rbf: radial basis function kernel, cca: ITQ+CCA initialisation.

$C = 300$ k-means anchor points, which is an identical to the number of anchor points used by KSH. For example, at 32 bits on CIFAR-10, GRH$_{rbf,full,cca}$ realises a 12% relative increase in mAP over the state-of-the-art KSH model, while GRH$_{rbf,300,cca}$ achieves a 4% increase. I note, however, that there is no significant difference between either my linear (GRH$_{lin,cca}$) or RBF parameterised models (GRH$_{rbf,1200,cca}$) and KSH on the larger NUS-WIDE dataset (Table 6.22). The former result is impactful, as the linear model (GRH$_{lin,cca}$) is much faster at training and prediction time than KSH (Section 6.3.3.2), but maintains the same level of accuracy on NUS-WIDE.

Finally my model does not explicitly enforce properties E_3 and E_4 of an effective hashcode as outlined by Weiss et al. (2008) and reviewed in Chapter 2, Section 2.6.1. Recall that property E_3 targets the *efficiency* of the bits and how they ideally should present a balanced partition of the input feature space so that not all N data-points end up in the same bucket. Property E_4, on the other hand, specifies that bits should *not be redundant* and that a good hashing model should therefore learn bits that are *pairwise independent*. ITQ+CCA learns hyperplanes that are pairwise orthogonal which is a relaxed version of the pairwise independence property, and the method also approximately preserves property E_3 due to the variance maximisation (Wang et al. (2012)). Other projection functions (PCAH, ITQ, AGH) I compared to in Section 6.3.3.5 also learn orthogonal hyperplanes courtesy of a matrix factorisation. This result suggests further experimentation to determine whether enforcing constraints E_3, E_4 is in fact necessary when learning effective hash functions. I leave this study to future work.

6.4 Discussion

In the first part of this chapter I introduced a new unimodal supervised projection function for hashing-based ANN search (Section 6.2). The hashing model is centered upon a *three step iterative* algorithm inspired by the Expectation-Maximisation (EM) algorithm of Dempster et al. (1977). Before running the model, I initialise a set of K-bit hashcodes for the training data-points by using an existing fingerprinting scheme such as Locality Sensitive Hashing (LSH). In the first step the model *regularises* (smooths) these training dataset hashcodes over a data-point *affinity graph* that specifies which training data-points are deemed similar and dissimilar according to human assigned judgments (Section 6.2.3). This first step has the effect of updating the hashcode for a data-point to be the average of the hashcodes of its nearest neighbours (before binarisation). The second step of the algorithm learns a set of K binary classifiers to predict the training dataset hashcodes with maximum margin (Section 6.2.4). This second step learns K hyperplanes that partition the input feature space in a manner that is consistent with the regularised hashcodes at that step. In other words, the learnt hyperplanes should be able to encode the training dataset to give hashcodes that are similar if not identical to the regularised hashcodes. As these hashcodes were previously regularised in Step A according to a graph built out of supervisory information, in this way we are indirectly able to infuse a degree of supervision into the hyperplane learning procedure. The learnt hyperplanes are then used to *re-label* the training dataset points with updated hashcodes which has the effect of flipping the bits of those data-points that could not be correctly classified in the second step (i.e. data-points that fell on the wrong side of one or more hyperplanes) (Section 6.2.5). These relabeled hashcodes are then fed back into the first step ready for the next iteration. These steps are then repeated for a fixed number of iterations. This three step procedure neatly allowed us to avoid an NP-hard optimisation problem involving the sign function. To the best of my knowledge the model presented in this chapter is the first projection function for hashing-based ANN search that uses graph regularisation as a means of integrating supervision into hyperplane learning.

In an extensive series of experiments on the task of query-by-example image retrieval I demonstrated the benefit of my approach. The evaluation of the algorithm was broken down into six hypotheses that were designed to isolate the effect of different aspects of the model. The findings resulting from the six experiments are highlighted below:

- Learnt hyperplanes result in more effective hashcodes than randomly placed hyperplanes. I found a significantly higher retrieval effectiveness arising from hashcodes generated from learnt hyperplanes versus those hashcodes generated from data-independent projection functions such as LSH and SKLSH. Random hyperplanes run the risk of partitioning regions of the space dense in related data-points, whereas learnt hyperplanes can be more effectively positioned based on supervision. Quantitative results supporting this claim can be found in Section 6.3.3.4, Tables 6.6-6.8.

- Hyperplanes learnt using supervision generate more effective hashcodes than hyperplanes learnt using an unsupervised but data-dependent matrix factorisation such as PCA or the Laplacian Eigenmap. This claim is supported by the results in Section 6.3.3.5, Tables 6.13-6.14.

- Regularising hashcodes over a data-point affinity graph is a more effective method of integrating supervision into the process of hyperplane learning than a Laplacian Eigenmap dimensionality reduction. Results relating to this claim are presented in Section 6.3.3.6, Tables 6.15-6.16.

- Initialising hashcodes with a supervised embedding can yield a higher retrieval effectiveness than a random initialisation. Nevertheless, a random initialisation resulted in an impressive, albeit lower, retrieval effectiveness compared to a supervised initialisation on both of the considered image datasets (CIFAR-10 and NUS-WIDE). Relevant results are presented in Section 6.3.3.7, Tables 6.17-6.18.

- Non-linear hypersurfaces induced by the radial basis function (RBF) kernel provide a more effective partitioning of the input feature space compared to linear hypersurfaces (hyperplanes). The complex non-linear decision boundaries created by the RBF kernel ensure more related data-points end up within the same partitioned regions of the input feature space. The more effective partitioning ultimately led to more discriminative hashcodes and a higher observed retrieval effectiveness, at the expense of efficiency. Relevant results are presented in Section 6.3.3.8, Tables 6.19-6.20.

While I am encouraged with the retrieval performance of my graph regularised projection function I note that it is currently restricted to unimodal hashing in which both the query and database are encoded with the same feature descriptor (e.g. GIST Oliva

and Torralba (2001)). As argued in Chapter 2, Section 2.6.5 many modern datasets are *multi-modal* in nature, being associated with multiple different feature descriptors. It would be particularly useful if I could pose a query encoded using one feature descriptor (e.g. TF-IDF for a textual query) and match the query against a database encoded with another feature descriptor (e.g. GIST of images). I show in Section 6.5 how my projection function can be extended to obtain a state-of-the-art *cross-modal* projection function which generates similar binary hashcodes for similar data-points in two different modalities.

6.5 Extending the Model to Cross Modal Hashing

As touched upon in my review of previously related research in Chapter 2, the majority of existing hashing research has focused on generating binary codes for data-points within the same modality (feature space), for example, a text query executed against a database consisting of textual documents. However, it is frequently the case that similar data-points exist in different modalities, for example a Wikipedia page discussing Einstein and an associated image of the scientist. An interesting research question is whether an effective hashing scheme can be constructed to learn hashcodes that are also similar *across* disparate modalities - in this case the Einstein Wikipedia article will ideally be assigned a similar hashcode to the relevant embedded image. Hashing methods that effectively bridge the cross-modal gap will enable hashing-based ANN search to be expanded to cross-modal datasets.

In this section I show how it is possible to extend my unimodal graph regularised projection function introduced earlier in this chapter to hash data-points in two different modalities. Cross-modal hashing research has undoubtedly received increased interest over the past several years due to the recent emergence of large freely available cross-modal datasets from sources such as Flickr[7] (Kumar and Udupa (2011); Bronstein et al. (2010); Zhen and Yeung (2012); Song et al. (2013); Rastegari et al. (2013)). Existing cross-modal hashing schemes seek to jointly preserve the within-modality and between-modality similarities of related data-points in a shared Hamming space. As I discussed in Chapter 2, Section 2.6.5 this requirement is frequently solved by learning two sets of K hyperplanes that partition each input feature space into buckets in a manner that yields similar hashcodes for similar data-points both within and across the two modalities.

[7] http://www.flickr.com

6.5.1 Problem Definition

Let $\mathbf{C} = \{(\mathbf{a}_i, \mathbf{v}_i) : i = 1 \ldots N_{trd}\}$ denote a collection of N_{trd} annotated images. Each image is represented by two components: the annotation $\mathbf{a}_i$, and the visual descriptor $\mathbf{v}_i$. The annotation $\mathbf{a}_i$ is a vector over textual features. Visual descriptor $\mathbf{v}_i$ is a vector of real-valued visual features. Our goal is to learn a pair of hash functions f, g that map annotations and visual descriptors into binary hashcodes consisting of K bits. I impose two constraints on my hash functions: **(i)** the annotation hashcode $f(\mathbf{a}_i)$ should be similar to the visual hashcode $g(\mathbf{v}_i)$ of the same image so that both feature descriptors will end up in the same hashtable bucket; and **(ii)** the annotation hashcodes $f(\mathbf{a}_i)$ and $f(\mathbf{a}_j)$ should be similar whenever images i and j are considered *neighbours*. The neighbourhood structure for the collection is dictated by an affinity matrix $\mathbf{S}$, where $S_{ij} = 1$ indicates that i and j are neighbours, and $S_{ij} = 0$ indicates they are not.

6.5.2 Overview of the Approach

My cross-modal graph regularised projection function is based on the unimodal graph regularised projection function introduced in Section 6.2. That projection function was restricted to a single modality, while in this section I propose an extension that learns a pair of hash functions across two separate modalities: text annotations $\mathbf{a}_i$ and visual descriptors $\mathbf{v}_i$. The hash functions f, g are based on K hyperplanes each: $\{\mathbf{w}_1 \ldots \mathbf{w}_K\}$ for the space of words and $\{\mathbf{u}_1 \ldots \mathbf{u}_K\}$ for the space of visual features. The hyperplane $\mathbf{w}_j$ is used to assign the j'th bit in the annotation hashcode, while $\mathbf{u}_j$ determines the j'th bit in the visual hashcode. I initialise all hyperplanes randomly using unimodal LSH, and iteratively perform the following three steps: **(1) Regularisation**, where the hashcodes $\{\mathbf{b}_1 \ldots \mathbf{b}_N\}$ are made more consistent with the affinity matrix $\mathbf{S}$ and **(2) Partitioning**, where we adjust the hyperplanes $\mathbf{w}_j, \mathbf{u}_j$ to be consistent with the j'th bit of the hashcodes from step (2). Adjusting the visual hyperplanes based on the annotation bits is how I form the necessary cross-modal bridge[8]. **(3) Prediction**, the hyperplanes $\{\mathbf{w}_1 \ldots \mathbf{w}_K\}$ are then used to assign hashcodes $\{\mathbf{b}_1 \ldots \mathbf{b}_N\}$ to the training images which are then fed back into Step A ready for the next iteration. Pseudocode describing the salient parts of my model is provided in Algorithm 10. The key difference between GRH is shown on Lines 8-9, where the annotation space bits are used to learn hyperplanes in the annotation and visual feature spaces.

[8] Adjusting the visual hyperplanes based on annotation bits, and annotation hyperplanes based on visual bits is also possible, but is left to future work.

I note here that my model bears some similarities with the Predictable Dual-View Hashing (PDH) projection function of Rastegari et al. (2013). I covered PDH in detail in Chapter 2, Section 2.6.5.4 and previously gave a pseudocode specification of the PDH model in Algorithm 5 on page 88. In a nutshell PDH also employs an Expectation-Maximisation (EM)-like iterative learning scheme with a max-margin formulation to refine the positioning of the hashing hyperplanes within both feature spaces. In contrast to my model, PDH integrates supervision into the hyperplane learning by solely relying on initialising the hashcodes with Canonical Correlation Analysis (CCA). I previously described CCA in detail as part of my description of the ITQ+CCA model in Chapter 2, Section 2.6.4.1. CCA finds two sets of K hyperplanes, one set of K hyperplanes for each feature space, that result in projections that are maximally correlated for related data-points across the two modalities. PDH also explicitly seeks to induce a pairwise independence between the hashcode bits through the solution of a graph Laplacian eigenvalue problem after each iteration. In a different manner to PDH, I do not seek to enforce bit independence and I also do not rely on an initial CCA initialisation to integrate the supervisory signal into learning algorithm. Instead my proposed cross-modal hashing model eliminates the need to solve either eigenvalue system, relying on graph regularisation to enforce the data-point must-link and cannot-link constraints. I show in this section that my model can reach a higher retrieval effectiveness than PDH while also maintaining the attractive advantage of being more efficient at training time.

6.5.2.1 Initialisation

I start by assigning a K-bit binary hashcode $\mathbf{b}_i \in \mathbb{R}^K$ to each training image i. Each of the K bits in $\mathbf{b}_i$ is based on a dot product between the image annotation $\mathbf{a}_i \in \mathbb{R}^{D_x}$ and one of the hyperplanes $\left\{\mathbf{w}_1 \in \mathbb{R}^{D_x}, \ldots, \mathbf{w}_K \in \mathbb{R}^{D_x}\right\}$:

$$\mathbf{b}_i = f(\mathbf{a}_i) = sgn\left[\mathbf{w}_1^\intercal \mathbf{a}_i \ldots \mathbf{w}_K^\intercal \mathbf{a}_i\right] \tag{6.7}$$

At test time, hashcodes of visual features $g(\mathbf{v}_i)$ can be computed in the same manner, but using the visual hyperplanes $\left\{\mathbf{u}_1 \in \mathbb{R}^{D_z}, \ldots, \mathbf{u}_K \in \mathbb{R}^{D_z}\right\}$.

6.5.2.2 Step A: Regularisation

The aim of this step is to make the hashcodes more consistent with the affinity matrix $\mathbf{S}$. Specifically, whenever images i and j are neighbours, we would like the hashcodes

Algorithm 10: REGULARISED CROSS MODAL HASHING (RCMH)

Input: Annotation descriptors $\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D_x}$, Visual descriptors $\mathbf{Z}_{trd} \in \mathbb{R}^{N_{trd} \times D_z}$, adjacency matrix $\mathbf{S} \in \{0,1\}$, degree matrix $\mathbf{D} \in \mathbb{Z}_+$, interpolation parameter $\alpha \in [0,1]$, number of iterations $M \in \mathbb{Z}_+$

Output: Hyperplanes $\{\mathbf{w}_1 \ldots \mathbf{w}_K\}$, $\{\mathbf{u}_1 \ldots \mathbf{u}_K\}$, biases $\{t_1 \ldots t_K\}$, $\{o_1 \ldots o_K\}$

Initialise $\mathbf{B}_0 \in \{0,1\}^{N_{trd} \times K}$ via LSH or ITQ+CCA from $\mathbf{X}$
$\mathbf{B}_0 = \text{sgn}(\mathbf{B}_0 - \frac{1}{2})$
$\mathbf{B} = \mathbf{B}_0$
for $m \leftarrow 1$ ***to*** M **do**
  $\mathbf{B} = \text{sgn}\left(\alpha \mathbf{S}\mathbf{D}^{-1}\mathbf{B} + (1-\alpha)\mathbf{B}_0\right)$
  for $k \leftarrow 1$ ***to*** K **do**
    $\mathbf{b}_k = \mathbf{B}(:,k)$
    Train SVM_k^x with $\mathbf{b}_k$ as labels, training dataset $\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D_x}$
    Train SVM_k^z with $\mathbf{b}_k$ as labels, training dataset $\mathbf{Z}_{trd} \in \mathbb{R}^{N_{trd} \times D_z}$
    Obtain hyperplanes $\mathbf{w}_k, \mathbf{u}_k$ and biases t_k, o_k
  end
  $B_{ik} = \text{sgn}(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k)$ for $i = \{1 \ldots N_{trd}\}$ and $k = \{1 \ldots K\}$
end
return $\{\mathbf{w}_k\}_{k=1}^K, \{\mathbf{u}_k\}_{k=1}^K, \{t_k\}_{k=1}^K, \{o_k\}_{k=1}^K$

$\mathbf{b}_i$ and $\mathbf{b}_j$ to be similar in terms of their Hamming distance. I achieve this by interpolating the hashcode of image i with the hashcodes of all *neighbouring* images j for which $S_{ij} = 1$. This is computed using Equation 6.1 introduced in the context of my unimodal projection function and repeated in Equation 6.8 for reading convenience. I show this approach intuitively in Figure 6.11. Shown are five images $a \ldots e$ with their initial hashcodes (K=2 bits for this example). The lines between images reflect the neighbourhood structure encoded in the affinity matrix $\mathbf{S}$. Image d has a hashcode (-11), but its neighbours b, c, e have hashcodes (-1-1), (11) and (1-1) respectively. The arrow beside the initial hashcode (-11) of image d shows the effect of Equation 6.8: its hashcode changes to (1-1), which is more consistent with neighbouring hashcodes (on average).

$$\mathbf{B}_m \leftarrow \text{sgn}\left(\alpha\, \mathbf{S}\mathbf{D}^{-1}\mathbf{B}_{m-1} + (1-\alpha)\mathbf{B}_0\right) \tag{6.8}$$

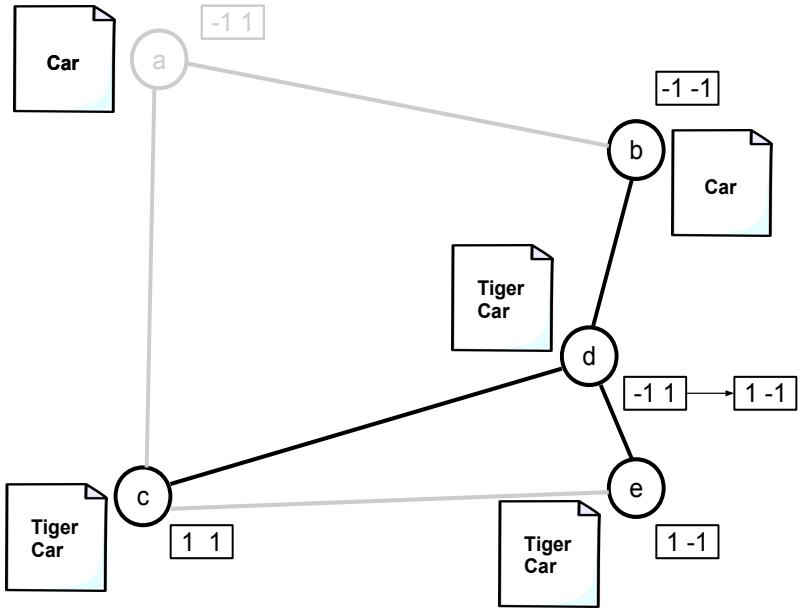


Figure 6.11: Regularisation step: the hashcode for annotation node d is updated to be more similar with its neighbours (c,b,e).

6.5.2.3 Step B: Partitioning

In this step I re-estimate the hyperplanes $\{\mathbf{w}_1 \ldots \mathbf{w}_K\}$ and $\{\mathbf{u}_1 \ldots \mathbf{u}_K\}$ to make them consistent with the regularised hashcodes from Step A of the algorithm. For each bit $j = \{1 \ldots K\}$, I treat the values $\{b_{1j} \ldots b_{N_{trd}j}\}$ as the training labels. Specifically, if $b_{ij} = 1$ then the annotation vector $\mathbf{a}_i$ constitutes a positive example for the hyperplane $\mathbf{w}_j$, and the visual vector $\mathbf{v}_i$ is a positive example for $\mathbf{u}_j$. If $b_{ij} = -1$ then $\mathbf{a}_i$ and $\mathbf{v}_i$ are negative examples for $\mathbf{w}_j$ and $\mathbf{u}_j$. Each hyperplane is learned using `liblinear` Fan et al. (2008) to maximise the margin between positive and negative examples. The approach is illustrated in Figure 6.12. I show five images $a \ldots e$ in two sets of coordinates: the word space on the top and the visual feature-space on the bottom. Each image is associated with a 2-bit hashcode, and each bit is used to learn a maximum-margin hyperplane that bisects the corresponding space. For example, the first bit has value -1 for images a, b and value 1 for images c, d, e, giving rise to hyperplanes $\mathbf{w}_1$ and $\mathbf{u}_1$, shown as dark lines on the top and the bottom parts of Figure 6.12. Note that $\mathbf{w}_1$ and $\mathbf{u}_1$ look very different, because they are defined over two completely different modalities: words on the top and visual features on the bottom. Similarly, the second bit results in the hyperplanes $\mathbf{w}_2$ and $\mathbf{u}_2$.

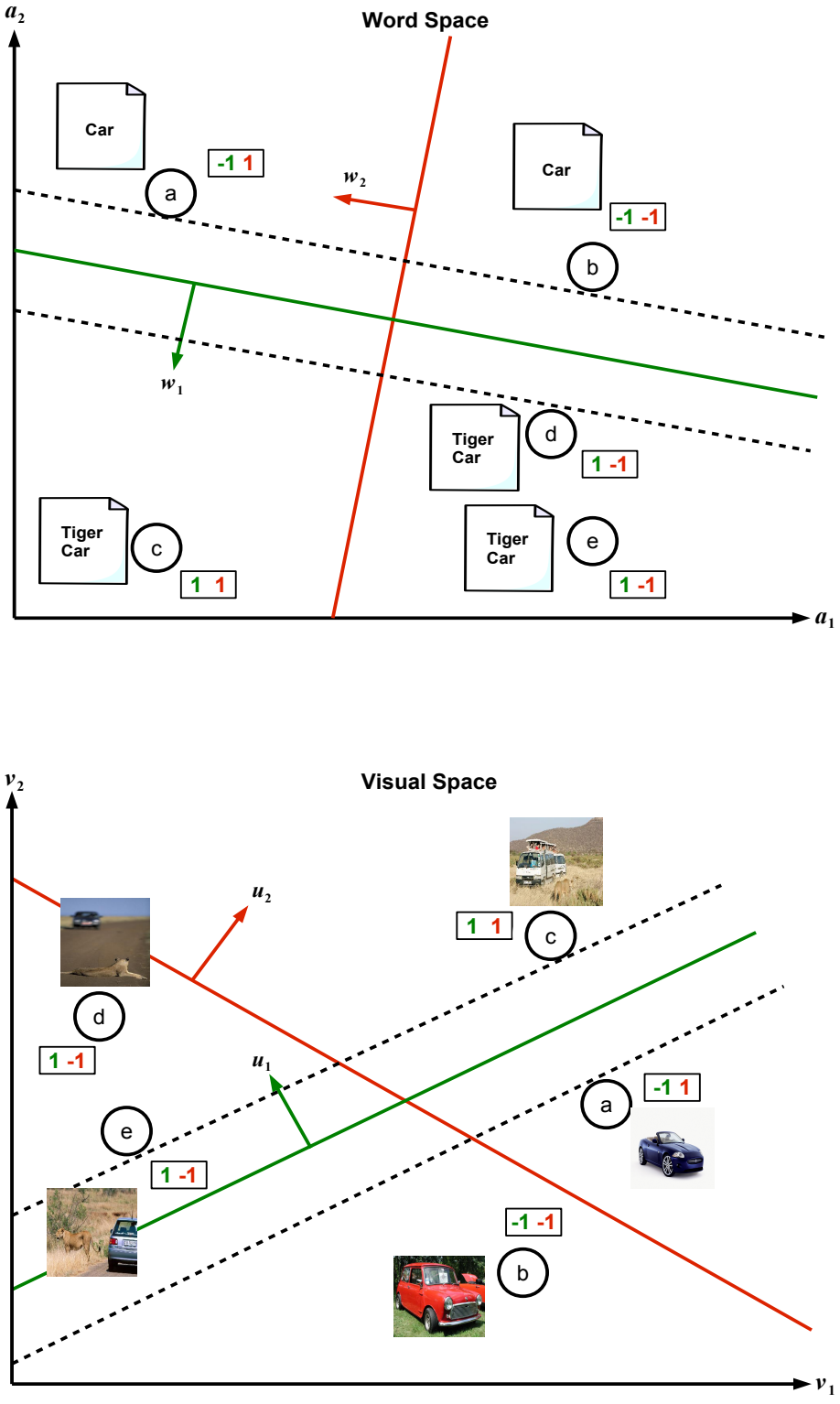


Figure 6.12: Partitioning step: hyperplanes are learnt in the annotation (top) and visual (bottom) space using annotation bits as labels. Hyperplanes in both feature spaces are positioned in such a way that nodes with the same letter are assigned the same bits in both feature spaces.

6.5.2.4 Step C: Prediction

In the final step of the current iteration the estimated hyperplane normal vectors for the annotation space $\{\mathbf{w}_1 \ldots \mathbf{w}_K\}$ are used to re-label the annotation data-points as given in Equation 6.9

$$B_{ik} = \text{sgn}(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k) \text{ for } i = \{1 \ldots N_{trd}\} \text{ and } k = \{1 \ldots K\} \tag{6.9}$$

6.5.3 Iteration and Constraints

I repeat steps A,B,C for a small number of iterations M. In this section, I briefly describe how the steps enforce the two constraints I imposed on my hash functions in Section 6.5.1. Constraint (i) is enforced in Step B of the algorithm, when I use the same bit values b_{ij} as targets for the word hyperplanes $\mathbf{w}_j$ and visual hyperplanes $\mathbf{u}_j$. Any image i will either be a positive example for both hyperplanes, or it will be negative for both, so at test time we can expect $\mathbf{w}_j^\intercal \mathbf{a}_i$ to yield the same bit value as $\mathbf{u}_j^\intercal \mathbf{v}_i$. Constraint (ii) is enforced in Step A of my procedure, where the hashcode for image i is moved towards the centroid hashcode of its neighbours. The centroid (before it is binarised) is a point that minimizes aggregate Euclidean distance to the neighbours, so after Step C hashcodes $\{\mathbf{b}_1 \ldots \mathbf{b}_{N_{trd}}\}$ are expected to be more consistent with the neighbourhood structure $\mathbf{S}$.

6.6 Experimental Evaluation

6.6.1 Experimental Configuration

I evaluate the cross-modal projection function on two publicly available benchmark datasets: Wiki and NUS-WIDE, both of which were described in Chapter 3, Section 3.2.2. As for my unimodal experiments the ground truth nearest neighbours are based on the semantic labels supplied with both datasets, that is, if an image and a document share a class in common they are regarded as true neighbours (Zhen and Yeung (2012); Song et al. (2013)). Following previous work (Zhen and Yeung (2012); Song et al. (2013)) I randomly select 20% (Wiki) and 1% (NUS-WIDE) of the data-points as queries with the remainder forming the database over which my retrieval experiments are performed. I also randomly sample 20% (Wiki) and 1% (NUS-WIDE) of the data-points from the database ($\mathbf{X}_{db} \in \Re^{N_{db} \times D_x}$, $\mathbf{Z}_{db} \in \mathbb{R}^{N_{db} \times D_z}$) to form the training dataset

Partition	Wiki	NUS-WIDE
Test queries (N_{teq})	574	1,866
Validation queries (N_{vaq})	574	1,866
Validation database (N_{vad})	1,144	18,666
Training database (N_{trd})	574	1,866
Test database (N_{ted})	2,292	184,711

Table 6.23: Literature standard splitting strategy partition sizes for the experiments in this chapter. This breakdown is based on the standard splitting strategy introduced in Chapter 3, Section 3.5.1.

($\mathbf{X}_{trd} \in \Re^{N_{trd} \times D_x}$, $\mathbf{Z}_{trd} \in \mathbb{R}^{N_{trd} \times D_z}$) to learn the hash functions. The exact division of the datasets into the different partitions is given in Table 6.23. The improved splitting strategy is not used here as it was found to have little effect on the retrieval results in Section 6.3.3.4, Table 6.6.

My model is evaluated on two cross-modal retrieval tasks: 1) *Image query vs. text database:* an image is used to retrieve the most related text in the text database; 2) *Text query vs. image database:* a text query is used to retrieve the most similar images from the image database. Retrieval effectiveness is again measured using the familiar *Hamming ranking* evaluation paradigm: binary codes are generated for both the query and the database items and the database items are then ranked in ascending order of the Hamming distance. I use these ranked lists to compute mean average precision (mAP) and all reported results are the average over ten random query/database partitions. The parameter optimisation strategy is the same as described for my unimodal projection function in Section 6.3.2. The experimental configuration used in this section is summarised in Table 6.24 and is designed to be identical to previously published cross-modal hashing research (Zhen and Yeung (2012), Song et al. (2013)) so that my reported figures are directly comparable.

The baselines in my experimental evaluation constitute five state-of-the-art cross-modal projection functions recently proposed in the learning to hash literature. Concretely I compare my model to Cross View Hashing (CVH) (Kumar and Udupa (2011)), Cross Modal Semi-Supervised Hashing (CMSSH) (Bronstein et al. (2010)), Co-Regularised Hashing (CRH) (Zhen and Yeung (2012)), Inter-Media Hashing (IMH) (Song et al. (2013)) and Predicable Dual-View Hashing (PDH) (Rastegari et al. (2013)). I introduced and discussed these five baseline cross-modal hashing models in consider-

Parameter	Setting	Chapter Reference
Groundtruth Definition	Class labels	Chapter 3, Section 3.3.2
Evaluation Metric	mAP	Chapter 3, Section 3.6.4
Evaluation Paradigm	Hamming Ranking	Chapter 3, Section 3.4
Random Partitions	10	Chapter 3, Section 3.5
Number of Bits (K)	24,48,64	Chapter 2, Section 2.4

Table 6.24: Configuration of the main experimental parameters for the cross-modal hashing results.

able detail in my review in Chapter 2, Section 2.6.5. Taken together these five models form a strong set of baselines for comparison.

6.6.2 Experimental Results

The cross-modal retrieval results are presented in Tables 6.25-6.26. I observe that the proposed model (RCMH) outperforms the baseline projection functions on both datasets and across all hashcode lengths. For example, for image-text retrieval, my model outperforms the most strongly performing baseline (PDH) by a substantial 16% relative mAP at 24 bits on the Wiki dataset and 9% on the NUS-WIDE dataset. I test the statistical significance of the gains in mAP versus the PDH model using a Wilcoxon signed rank test on the mAP scores resulting from each random test query/database partition. This test highlights that my model is significantly better ($p < 0.01$) at both text-image and image-text cross-modal retrieval than the second best model PDH. I further present two example qualitative cross-modal retrieval results in Figures 6.13-6.14.

The fact that my model achieves the highest retrieval effectiveness is an encouraging result given that it is devoid of a computationally expensive eigendecomposition step used by most baselines. GRH instead relies on regularising the hashcodes over an adjacency graph to maintain the neighbourhood structure. This suggests yet again that regularising hashcodes over an adjacency graph is more effective for hashing hypersurface learning than solving an eigenvalue problem. I previously presented evidence to this effect in the context of my unimodal projection function in Section 6.3.3.6. Compared to PDH, which has a training time complexity of $O(MN_{trd}D_xK + MN_{trd}D_zK + MKN_{trd}^2)$, my model is $O(MN_{trd}D_xK + MN_{trd}D_zK + MSK)$. I compare the timing results of two baselines (PDH, CMSSH) to my own model in Table 6.27. It is ap-

Task	Method	Hashcode Length		
		24 bits	**48 bits**	**64 bits**
Image Query vs. Text Database	CRH	0.1632	0.1752	0.1698
	CVH	0.1570	0.1519	0.1538
	CMSSH	0.1439	0.1501	0.1420
	IMH	0.1881	0.1892	0.1897
	PDH	0.2109	0.2186	0.2266
	RCMH	**0.2439▲▲**	**0.2463▲▲**	**0.2590▲▲**
Text Query vs. Image Database	CRH	0.1266	0.1239	0.1267
	CVH	0.1284	0.1176	0.1185
	CMSSH	0.1119	0.1123	0.1124
	IMH	0.1507	0.1514	0.1491
	PDH	0.1790	0.1860	0.1902
	RCMH	**0.2066▲▲**	**0.1918▲▲**	**0.2201▲▲**

Table 6.25: mAP scores for Wiki ($T = 574$). ▲▲ indicates statistical significance versus PDH (Wilcoxon signed rank test, p-value < 0.01).

parent from these results that RCMH requires approximately 9% of the training time of PDH and 7% of CMSSH, while having a similar sub-second prediction (encoding) time. My proposed model is therefore both faster to train and achieves more accurate nearest neighbour search results on cross-modal datasets compared to state-of-the-art baselines.

6.7 Discussion

In the second part of this chapter I extended the unimodal projection function introduced in Section 6.2 so that the model was able to assign similar hashcodes to similar data-points existing in disparate feature spaces, namely textual annotations and visual descriptors. Surprisingly the manner in which this cross-modal bridge was achieved was relatively straightforward and involved only minor adjustments to the unimodal algorithm. Specifically I retained the iterative multi-step scheme by firstly initialising the *annotation descriptor* hashcodes using LSH (Section 6.5.2.1). In the first step I then regularised these annotation hashcodes using the adjacency graph (Section 6.5.2.2). The second step of the multi-step algorithm is the step in which I form the cross-modal

Task	Method	Hashcode Length		
		24 bits	**48 bits**	**64 bits**
Image Query vs. Text Database	CRH	0.3536	0.3539	0.3588
	CVH	0.3397	0.3436	0.3412
	CMSSH	0.3429	0.3386	0.3382
	IMH	0.4022	0.4019	0.4040
	PDH	0.4217	0.4245	0.4272
	RCMH	**0.4605▲▲**	**0.4719▲▲**	**0.4649▲▲**
Text Query vs. Image Database	CRH	0.3495	0.3427	0.3481
	CVH	0.3394	0.3435	0.3410
	CMSSH	0.3429	0.3377	0.3492
	IMH	0.3926	0.3960	0.3997
	PDH	0.4053	0.4081	0.4096
	RCMH	**0.4325▲▲**	**0.4380▲▲**	**0.4350▲▲**

Table 6.26: mAP scores for NUS-WIDE ($T = 1866$). ▲▲ indicates statistical significance versus PDH (Wilcoxon signed rank test, p-value < 0.01).

	Train Complexity	Time	Test Complexity	Time
RCMH	$O(MN_{trd}D_xK + MN_{trd}D_zK + MSK)$	0.336	$O(N_{teq}DK)$	0.003
PDH	$O(M^{'}N_{trd}D_xK + M^{'}N_{trd}D_zK + M^{'}KN_{trd}^2)$	3.715	$O(N_{teq}DK)$	0.003
CMSSH	$O(KD_x^2D_z + KD_xD_z^2 + KD_z^2)$	5.172	$O(N_{teq}DK)$	0.003

Table 6.27: Training and testing time (seconds) on the Wiki dataset at 24 bits. RCMH is parameterised with $M = 1, M^{'} = 5$ and $N_{trd} = 574, N_{teq} = 574, D_x = 128, D_z = 10$. The timing results were recorded on an otherwise idle Intel 2.7GHz, 16Gb RAM machine and averaged over ten runs. All models are implemented in the same software stack (Matlab).

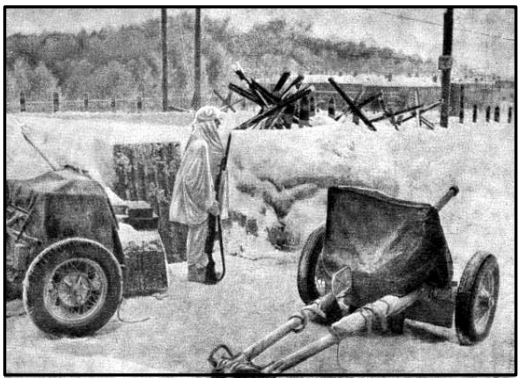
bridge and it is here where the unimodal and cross-modal versions of my projection function differ. This bridge was achieved by simply learning a *second set* of K binary classifiers to predict the *annotation hashcode* bits using the *visual descriptors* as features (Section 6.5.2.3). In this way the hypersurfaces in the visual feature space are positioned to predict the annotation hashcode bits with maximum margin. When used to generate hashcodes for the visual descriptors these visual hyperplanes are therefore expected to generate hashcodes that are broadly consistent with the hyperplanes in

the annotation feature space. In a set of experiments in Section 6.6.2, I demonstrated that this was not only the case, but in fact my cross-modal adaptation of the graph regularised projection function was shown to obtain state-of-the-art retrieval effectiveness on standard image datasets and against a set of strong baseline hashing models. Furthermore my model only required a fraction of the training time of competitive baselines (Table 6.27). In my analysis of the experimental results I suggested that the lack of an eigendecomposition step in my model was responsible for both the more efficient training time and the higher observed retrieval effectiveness.

6.8 Conclusions

In this chapter I developed a new supervised projection function for hashing-based ANN search (Section 6.2). In an extensive set of experiments, I demonstrated the effectiveness of the model for unimodal query-by-image retrieval. In a second contribution in Section 6.5, I then extended the model to hash cross-modal (text-image) data-points that consisted of two distinct feature descriptors. Both the unimodal and cross-modal variants of the algorithm consisted of two main steps performed iteratively. In the first step, training dataset hashcode bits are set to be the average of their nearest neighbours as defined by an adjacency graph. This step is known as *graph regularisation* and formed a key part of the model proposed in this chapter given that it enforced the important constraint that similar data points (as defined by the available supervision) should have similar hashcodes. In comparison to previous research which tend to rely on solving an eigenvalue system to integrate supervision into the hyperplane learning procedure, I instead investigated the applicability of graph regularisation to achieve the same objective. In the second step the regularised hashcodes form the labels for a set of binary classifiers, which has the effect of evolving the positioning of the hashing hypersurfaces to separate opposing bits $(0, 1)$ with maximum margin. Both steps are repeated for a fixed number of iterations. The learnt hypersurfaces can then be used to generate the hashcodes for novel query data-points. The proposed graph regularised projection function combined simplicity of implementation, competitive training time and state-of-the-art retrieval effectiveness both for unimodal and cross-modal applications. I believe these factors make the model one of the first supervised projection functions for hashing that has the potential to scale to truly massive unimodal and multi-modal datasets prevalent in the modern world.

In Chapter 7 I will demonstrate how the supervised projection function proposed



Neil Hamilton Fairley

From Wikipedia, the free encyclopedia

Brigadier **Sir Neil Hamilton Fairley** KBE CStJ FRACP FRCP FRCPE FRS[1] (15 July 1891 – 19 April 1966) was an Australian physician, medical scientist, and army officer; who was instrumental in saving thousands of Allied lives from malaria and other diseases.

A graduate of the University of Melbourne, Fairley joined the Australian Army Medical Corps in 1915. He investigated an epidemic of meningitis that was occurring in Army camps in Australia. While with the 14th General Hospital in Cairo, he investigated schistosomiasis (then known as bilharzia) and developed tests and treatments for the disease. In the inter-war period he became renowned as an expert on tropical medicine.

Fairley returned to the Australian Army during the Second World War as Director of Medicine. He played an important role in the planning for the Battle of Greece, convincing the British Commander-in-Chief, General Sir Archibald Wavell to alter his campaign plan to reduce the danger from malaria. In the South West Pacific Area, Fairley became responsible for co-ordinating the

Yellowstone fires of 1988

From Wikipedia, the free encyclopedia

The **Yellowstone fires of 1988** together formed the largest wildfire in the recorded history of Yellowstone National Park in the United States. Starting as many smaller individual fires, the flames quickly spread out of control with increasing winds and drought and combined into one large conflagration, which burned for several months. The fires almost destroyed two major visitor destinations and, on September 8, 1988, the entire park was closed to all non-emergency personnel for the first time in its history.[1] Only the arrival of cool and moist weather in the late autumn brought the fires to an end. A total of 793,880 acres (3,213 km^2), or 36 percent of the park was affected by the wildfires.[2]

Siege of Malakand

From Wikipedia, the free encyclopedia

The **Siege of Malakand** was the 26 July – 2 August 1897 siege of the British garrison in the Malakand region of colonial British India's North West Frontier Province.[8] The British faced a force of Pashtun tribesmen whose tribal lands had been bisected by the Durand Line,[9] the 1,519 mile (2,445 km) border between Afghanistan and British India drawn up at the end of the Anglo-Afghan wars to help hold the Russian Empire's spread of influence towards the Indian subcontinent.

The unrest caused by this division of the Pashtun lands led to the rise of Saidullah, a Pashtun fakir who led an army of at least 10,000[3][10] against the British garrison in Malakand. Although the British forces were divided amongst a number of poorly defended positions, the small garrison at the camp of Malakand South and the small fort at Chakdara were both able to hold out for six days against the much larger Pashtun army.

William Bostock

From Wikipedia, the free encyclopedia

Air Vice Marshal **William Dowling (Bill) Bostock**, CB, DSO, OBE (5 February 1892 – 28 April 1968) was a senior commander in the Royal Australian Air Force (RAAF). During World War II he led RAAF Command, the Air Force's main operational formation, with responsibility for the defence of Australia and air offensives against Japanese targets in the South West Pacific Area. His achievements in the role earned him the Distinguished Service Order and the American Medal of Freedom. General Douglas MacArthur described him as "one of the world's most successful airmen".

Battle of Lissa (1811)

From Wikipedia, the free encyclopedia

For the Battle of Lissa fought between Austria and Italy in 1866, see Battle of Lissa (1866).

The **Battle of Lissa** (sometimes called the **Battle of Vis**; French: *Bataille de Lissa*; Italian: *Battaglia di Lissa*; Croatian: *Viška bitka*) was a naval action fought between a British frigate squadron and a substantially larger squadron of French and Venetian frigates and smaller ships on 13 March 1811 during the Adriatic campaign of the Napoleonic Wars. The engagement was fought in the Adriatic Sea for possession of the strategically important island of Lissa (also known as Vis), from which the British squadron had been disrupting French shipping in the Adriatic. The French needed to control the Adriatic to supply a growing army in the Illyrian Provinces, and consequently dispatched an invasion force in March 1811 consisting of six frigates, numerous smaller craft and a battalion of Italian soldiers.

Figure 6.13: Example cross-modal retrieval result using the model proposed in this chapter. I pose an image query depicting a war scene and return the top five Wikipedia articles deemed to be relevant by my model. In this case four out of five of the articles are related to the theme of war (the Yellowstone fires are irrelevant to the query).

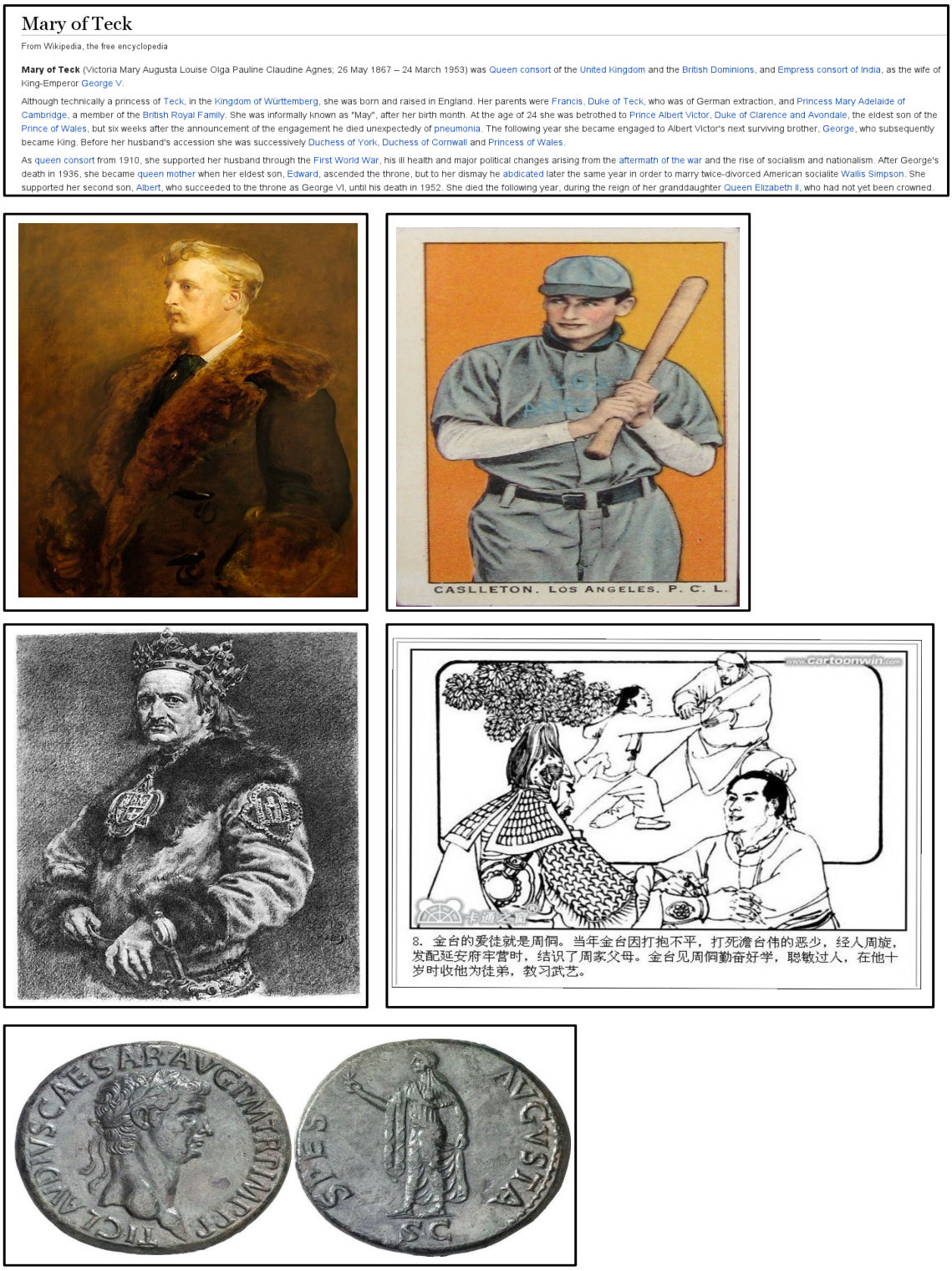


Figure 6.14: Example of text illustration using the cross-modal hashing model proposed in this chapter. I pose a text query regarding a member of the UK royal family (Mary of Teck) and the model finds the best five images to illustrate the text. Three out of five of the returned images are relevant to the general theme of royalty. The relevant images depict the Marquess of Lorne, coins from the era of Claudius the Roman Emperor and Jogaila Grand Duke of Lithuania.

in this chapter can be combined with the multiple threshold learning algorithms introduced in Chapters 4-5 to create one of the first hashing models for ANN search that is capable of learning multiple quantisation thresholds and task-specific hashing hypersurfaces. In effect this contribution transforms the hashcode generation pipeline of projection followed by quantisation from a process that relies on randomness and the associated asymptotic performance guarantees, to a pipeline that is now fully data-adaptive and capable of adjusting both the thresholds and hypersurfaces to the distribution of data.