Learning to Hash for Large Scale Image Retrieval

Sean J. Moran

Doctor of Philosophy

Institute for Language, Cognition and Computation

School of Informatics

University of Edinburgh

2016

Abstract

This thesis is concerned with improving the effectiveness of nearest neighbour search. Nearest neighbour search is the problem of finding the most similar data-points to a query in a database, and is a fundamental operation that has found wide applicability in many fields. In this thesis the focus is placed on hashing-based approximate nearest neighbour search methods that generate similar binary hashcodes for similar data-points. These hashcodes can be used as the indices into the buckets of hashtables for fast search. This work explores how the quality of search can be improved by *learning task specific* binary hashcodes.

The generation of a binary hashcode comprises two main steps carried out sequentially: *projection* of the image feature vector onto the normal vectors of a set of hyperplanes partitioning the input feature space followed by a *quantisation* operation that uses a single threshold to binarise the resulting projections to obtain the hashcodes. The degree to which these operations preserve the relative distances between the data-points in the input feature space has a direct influence on the effectiveness of using the resulting hashcodes for nearest neighbour search. In this thesis I argue that the retrieval effectiveness of existing hashing-based nearest neighbour search methods can be increased by learning the thresholds and hyperplanes based on the *distribution of the input data*.

The first contribution is a model for learning multiple quantisation thresholds. I demonstrate that the best threshold positioning is projection specific and introduce a novel clustering algorithm for threshold optimisation. The second contribution extends this algorithm by learning the optimal allocation of quantisation thresholds per hyperplane. In doing so I argue that some hyperplanes are naturally more effective than others at capturing the distribution of the data and should therefore attract a greater allocation of quantisation thresholds. The third contribution focuses on the complementary problem of learning the hashing hyperplanes. I introduce a multi-step iterative model that, in the first step, regularises the hashcodes over a data-point adjacency graph, which encourages similar data-points to be assigned similar hashcodes. In the second step, binary classifiers are learnt to separate opposing bits with maximum margin. This algorithm is extended to learn hyperplanes that can generate similar hashcodes for similar data-points in two different feature spaces (e.g. text and images). Individually the performance of these algorithms is often superior to competitive baselines. I unify my contributions by demonstrating that learning hyperplanes and thresholds as part of the same model can yield an additive increase in retrieval effectiveness.

Lay Summary

We are interested in finding similar images in large image databases, a task that is popularly known as image retrieval. In this task images are represented not as pixels, but as a feature vector consisting of a series of numbers that succinctly capture the salient properties of the image, such as the colour or texture distribution. We present the search system with the feature vector of a query image depicting the scene or object we wish to find in the database. The database then returns a set images most related to our image query. A simple way to solve this search problem is to compare the query image feature vector to the features of every image in the database. In this thesis I examine faster algorithms that permit nearest neighbours to be found without an exhaustive comparison. These algorithms convert an image feature vector into a *hashcode* that consists of bits i.e. a string of 0's and 1's. The hashcodes are more similar, that is share more bits in common, for images that describe the same scenes. This property allows the hashcodes to be used as pointers into the buckets of a hashtable, a standard Computer Science data structure that permits direct access to images that share the same hashcode as the query. We can therefore find related images while ignoring the majority of the images in the database.

This dissertation improves the quality of the hashcodes so that the hashcodes of similar images share more bits in common. We can imagine each image feature vector as representing a point in a high-dimensional "image space". A hashcode is then nothing more than a geometric identifier that points to the region occupied by a group of related images in this space. To generate a hashcode we divide up the space using hyperplanes (in two dimensions a hyperplane is a line). Each bit of the hashcode is determined by finding out on which sides of the hyperplanes each image feature vector lies: mathematically this involves computing the distances of an image feature vector from the hyperplanes followed by thresholding the resulting numbers to obtain binary bits. A bit is set to '0' if the image lies on one side of a given hyperplane and a '1' if it lies on the other side. This thesis introduces new algorithms for intelligently positioning the hyperplanes and thresholds in a way that encourages more related images to be within the same partitioned regions of the "image space". This goal is achieved by providing the algorithms with a small amount of human assigned supervision identifying which images should be considered similar and which images should be considered dissimilar. I find that my proposed algorithms facilitate the generation of high quality hashcodes that allow many more relevant images to be retrieved from the database.

Acknowledgements

This thesis would not have been possible without the guidance and valuable feedback of my principal supervisor Dr. Victor Lavrenko. I am deeply appreciative of the time Victor has invested in my PhD project and for all the interesting and in depth discussions we have had during my time as a research student. Directly experiencing Victor's approach to solving challenging problems and his innate ability to explain difficult concepts with incredible clarity have undoubtedly shaped me into both a much better researcher. There is no part of this thesis that has not directly benefited from Victor's expertise and experience.

I significantly benefited from the extensive computational resources Dr. Miles Osborne made available for my experimentation, and I am also grateful to Miles for providing me with the valuable practical experience of an enjoyable six month internship on the ReDiTes project during my PhD studies. I am thankful to Dr. Charles Sutton for serving on my yearly review committee and for his helpful advice on my research. I also greatly appreciate the considerable time and effort that my two examiners, Professor Robert Fisher and Professor Iadh Ounis, spent in reading and critiquing my thesis and for a pleasant and interesting discussion on my research during the Viva. Their feedback and comments have directly contributed to improving the work presented in this dissertation.

My family and friends have been instrumental in their encouragement throughout my doctoral studies. A special thank you to my parents, grandparents and two sisters, Siobhan and Claire, for their constant and unwavering support both during my time at Edinburgh University and throughout my life. I cannot express enough how grateful I am to you all. Dr. Christopher Town inspired me to follow my interest in Computer Vision at both MSc and PhD level through an enjoyable Cambridge University undergraduate dissertation and has provided valuable support throughout my higher education, in addition to an unforgettable trip to Sri Lanka. My appreciation goes to Aidan and Manfred for catching numerous spelling and grammar mistakes in earlier drafts of the thesis, and to Manfred for additionally proof-reading my conference papers and for inviting me to give a talk on my research at the Graz University of Technology, Austria. Thank you to Jess for all the special memories during my final year of study and for always being there for me. I look forward to that first trip to the Orkney Islands!

I had the privilege of attending several fantastic conferences during my time as a research student, all of which were located in many fantastic destinations around the world: from the green fields of Ireland and the serene lake-side beauty of Zürich to the arid Atacama desert of Chile, with its massive salt flats and jaw-dropping views of the Milky Way. Special thanks goes to the members and anonymous referees of SIGIR'13, ACL'13, ECIR'15, SIGIR'15 and SIGIR'16 for their numerous comments on my work and for funding my conference travel and registration fees, and to Google for hosting an enjoyable and informative PhD student conference. Finally, I am particularly grateful to the School of Informatics at the University of Edinburgh for providing financial assistance in the form of an EPSRC grant which has made the research reported in this thesis possible.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Sean J. Moran*)

To my family, to whom I am eternally grateful

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

The rapid growth of the World Wide Web (WWW) over the past two decades has brought with it a phenomenal increase in the amount of image, video and text based data being collected, stored and shared across the world. This phenomenon has been fuelled by the popularity of social media networks, cheap disk storage and the wide availability of Internet-enabled smartphones. For example it has been estimated that Facebook has in the order of 300 million images uploaded per day[1], YouTube receives 10 years worth of content per day[2] and there are now estimated to be well over 1 trillion web pages[3] in existence (Murphy (2012)). Figure 1.1 illustrates the explosive growth of images being uploaded onto popular social media websites and applications during the period 2005-2014. The trend towards real-time video sharing over the Internet with applications such as Periscope, involving a medium that is many times the size of individual images or documents, will severely exacerbate this torrent of data. In the near-term future the emergence of the Internet-of-Things (IoT) and Smart Cities promise to add further fuel to this fire, hinting at a connected society in which Internet linked sensors embedded in everyday objects, such as CCTV cameras and thermostats, produce an abundance of data that is automatically captured, stored and analysed so as to produce actionable insights for interested citizens and government stakeholders (Albakour et al. (2015)). The sheer scale of the data being produced around the world brings with it a need for computationally efficient algorithms that ensure the storage

[1]Velocity 2012: Jay Parikh, “Building for a Billion Users”

[2]`http://www.youtube.com/yt/press/en-GB/statistics.html`

[3]`http://googleblog.blogspot.co.uk/2008/07/we-knew-web-was-big.html`

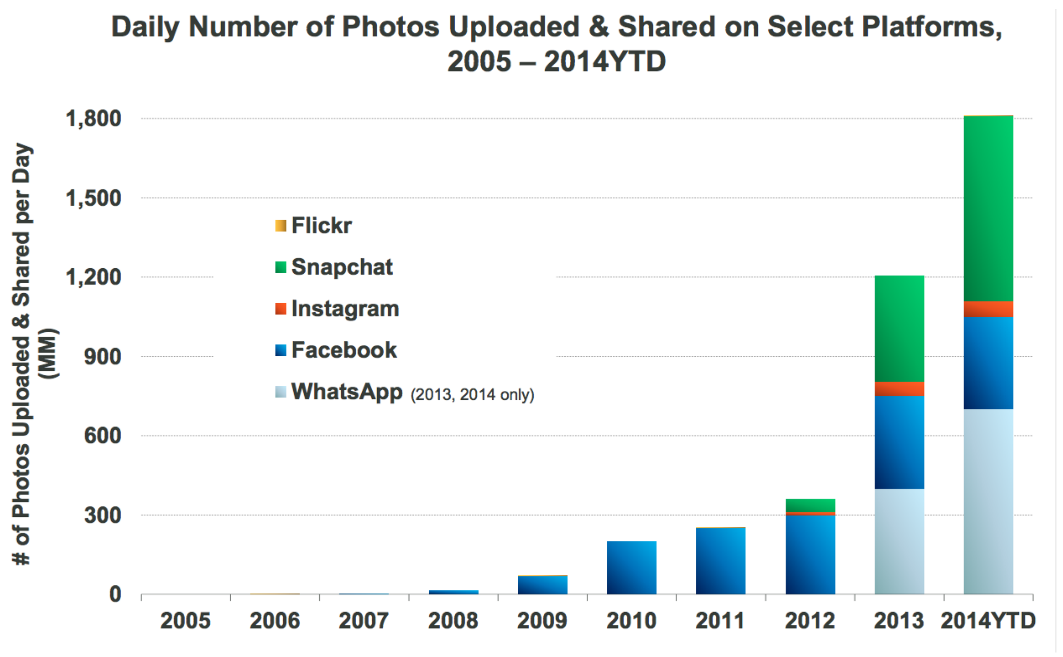


Figure 1.1: The amount of images being uploaded to popular social media websites (Facebook, Flickr) and mobile applications (Instagram, SnapChat, WhatsApp) has undergone a dramatic growth since 2005. Efficient algorithms for searching through such large image datasets are needed now more than ever. This chart has been copied directly from slide 62 of the talk *"Internet Trends 2014 - Code Conference"* given by the venture capitalist Mary Meeker of Kleiner Perkins Caufield Byers (KPCB): `http://www.kpcb.com/blog/2014-internet-trends` (URL accessed on 16/12/15).

requirements and processing overhead do not grow with the quantity of the data being produced.

In this thesis I focus on the problem of nearest neighbour (NN) search where the goal is to find the most similar data-point(s) to a query in a large database. Similarity is typically judged by representing the data-points as fixed dimensional vectors in a vector space and computing a distance metric such as the Euclidean or cosine distance. In this case data-points with a sufficiently low distance are judged to be nearest neighbours. Due to its generality and usefulness nearest-neighbour search finds application in many areas of Science, ranging from the field of Information Retrieval (IR) where we wish to find documents relevant to a query, to the problem of Genomic assembly in the field of Bioinformatics. The näive way to solve the nearest-neighbour search problem would be to compare the query to every data-point in the database, a method known as *brute-force search*. Brute-force search is only feasible in relatively small databases where performing the number of required comparisons between the data-points remains computationally tractable. Given the linear scaling of the query time with respect to the dataset size it is impossible to exhaustively search large-scale datasets consisting of millions to billions of data-points for nearest neighbours in a reasonable amount of time. This problem is compounded in the streaming data scenario where data-points need to be processed sequentially in real-time with potentially no end to the amount of incoming data. To efficiently find nearest-neighbours in large-scale datasets, algorithms are required that offer a query time that is independent of the dataset size.

Hashing-based approximate nearest neighbour (ANN) search methods are a popular class of algorithms that permit the nearest neighbours to a query data-point to be retrieved in constant time, independent of the dataset size. Hashing has proved to be an extremely useful method for ANN search over high-dimensional, large-scale datasets that are prevalent in the modern data-rich world. Hashing permits constant time search per query by condensing both the database and the query into fixed-length compact binary hashcodes or fingerprints. The hashcodes exhibit the neighbourhood preserving property that similar data-points will be assigned similar (low Hamming distance) hashcodes. Crucially, unlike cryptographic hash functions such as MD5 or SHA-1, the data-points need not be identical to receive matching hashcodes. Rather the degree of similarity between the hashcodes is a direct function of the similarity between the feature representation of the data-points. This property is ideal for my chosen task of image retrieval where we rarely wish to find only those images that are identical down to the pixel level. Most people would deem two images to be related even if the semantically equivalent objects (e.g. a tiger) depicted in both images are in widely different poses, and therefore the images have a completely different pixel consistency.

The aforementioned similarity preserving property enables the hashcodes to be used as the keys into the buckets of hashtables so that similar, but not necessarily identical, images will collide in the same buckets (Figure 1.2). This is a rather different use-case to the typical application of hashtables in Computer Science in which it is imperative to avoid collisions between non-identical data-points. In hashing-based ANN search we are actively encouraging collisions between similar data-points. The bucketing of the data-points drastically reduces the computational overhead of nearest neighbour search by reducing the number of comparisons that are required between the data-points: at query time we need only compare our query to those data-points colliding in the same buckets. There is no free lunch however as we pay for the reduced query time with a non-zero probability of failing to retrieve the closest nearest

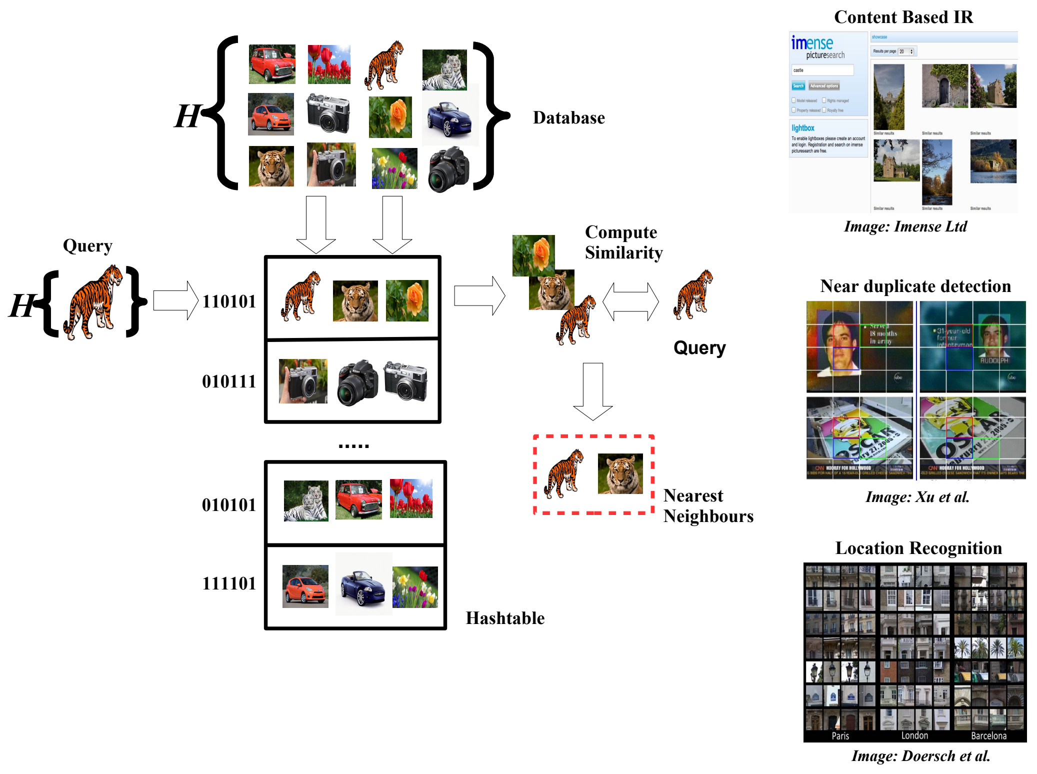


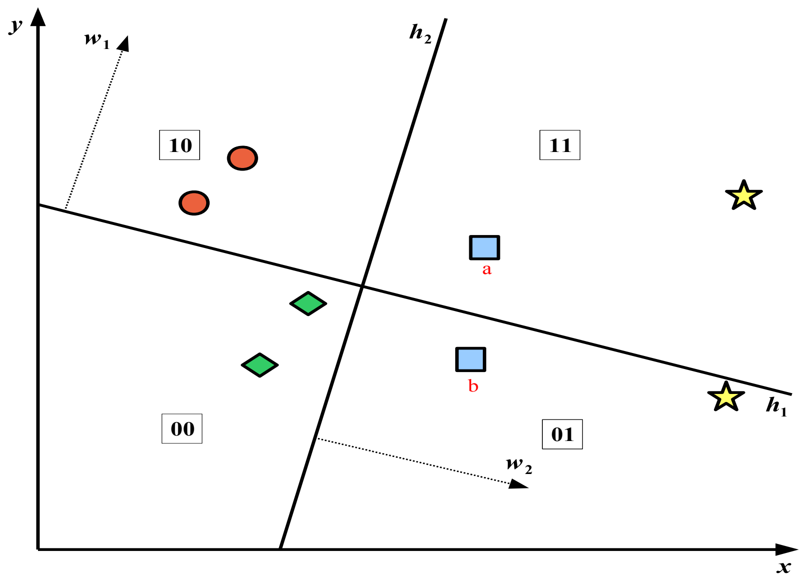
Figure 1.2: Nearest neighbour search with hashcodes. Similarity preserving binary codes generated by a hash function $\mathcal{H}$ can be used as the indices into the buckets of a hashtable for constant time search. Only those images that are in the same bucket as the query need be compared thereby reducing the size of the search space. The focus of this thesis is learning the hash function $\mathcal{H}$ to maximise the similarity of hashcodes for similar data-points. On the right-hand side I present examples of tasks for which nearest neighbour search has proved to be fundamental: from content-based information retrieval (IR) to near duplicate detection and location recognition. The three images on the right have been taken from Imense Ltd (`http://www.imense.com`) and Doersch et al. (2012); Xu et al. (2010); Grauman and Fergus (2013).

neighbours in the case where they happen to fall in different buckets. Nevertheless this quantifiable non-zero false negative probability turns out to be an acceptable trade-off in many application areas in which sub-optimal nearest neighbours can be almost as good as finding the exact nearest neighbour (Dean et al. (2013); Petrović et al. (2010)).

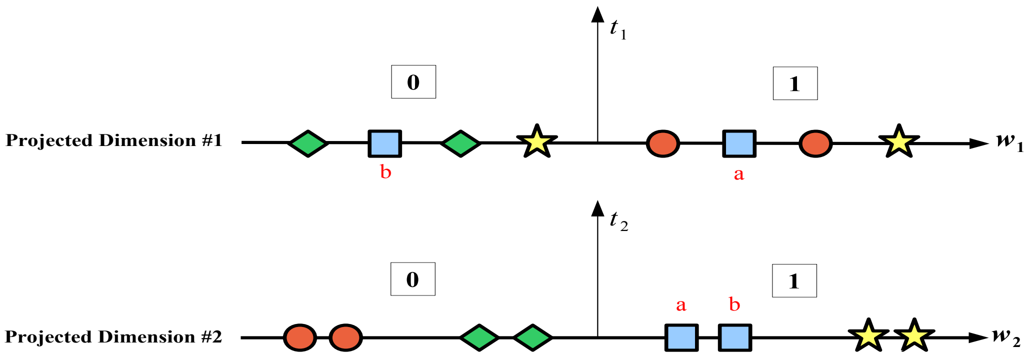
This thesis makes fundamental contributions to increasing the retrieval effectiveness of the core algorithmic processes underlying a well-known and widely applied method for hashing-based ANN search. I evaluate the effectiveness of these contributions on the task of content-based image retrieval, a signature problem encountered within the fields of IR and Computer Vision that is characterised by an abundance of data and the need for accurate and efficient search. Hashing-based ANN has also shown great promise in terms of efficient query processing and data storage reduction across a wide range of other interesting application areas within IR and Computer Vision. For example Petrović et al. (2010) present an efficient method for event detection in Twitter that scales to unbounded streams through a novel application of Locality Sensitive Hashing (LSH), a seminal randomised approach for ANN search (Indyk and Motwani (1998)). In the streaming data scenario of Petrović et al. (2010) the $O(N)$ worst case complexity of inverted indexing is undesirable, motivating the use of LSH to maintain a constant $O(1)$ query time[4]. Hashing-based ANN has also proved particularly useful for search over dense and much lower dimensional (compared to text) feature vectors, such as GIST (Oliva and Torralba (2001)), that are commonly employed in the field Computer Vision. In one recent application of LSH within Computer Vision, similarity preserving hashcodes have been successfully used for fast and accurate detection of 100,000 object classes on just a single machine (Dean et al. (2013)).

The ANN search hashing models I will examine and build upon in this thesis all partition the input feature space into disjoint regions with a set of hypersurfaces, either linear (hyperplanes) or non-linear. In the case of linear hypersurfaces the polytope-shaped regions formed by the intersecting hyperplanes constitute the hashtable buckets (Figure 1.3). The hashtable key for a data-point is generated by simply determining which side of the hyperplanes the data-point lies. Depending on which side it falls a '0' or a '1' is appended to the hashcode for that data-point. By repeating this procedure for each hyperplane we can build up a hashcode for each data-point that is the same length as the number of hyperplanes partitioning the space. Intuitively, the hashcode can be thought of as an identifier that captures the geometric position of the data-points within the input feature space with each bit encoding the position of the data-point with respect to a given hyperplane. Algorithmically this hashcode generation procedure can be accomplished in two separate steps performed in a pipeline: *projection* followed by *quantisation*. This procedure is illustrated with a toy example in Figure 1.3. Projection involves a dot product of the feature vector representation of a data-point onto

[4]This is only true if we ignore the hashing cost (cost of generating the hashcode) and assume that each database data-point goes into its own hashtable bucket. In practice the LSH computational cost for a single hashtable and a single data-point is a sum of the hashing cost ($O(KD)$), lookup cost ($O(1)$) and the candidate test cost ($O(ND/2^K)$), where K is the hashcode length and assuming a uniform distribution of data-points to buckets. Observe that there is a trade-off between the hashing cost and the candidate test cost, both of which are dependent on K. For example, in the situation where the data-points are evenly distributed into their own hashtable bucket ($N = 2^K$), the total computational cost for LSH is actually sub-linear ($O(D \log N)$).



(a) **Projection**



(b) **Quantisation**

Figure 1.3: The projection and quantisation operations. In Figure (a) a 2D space is partitioned with two hyperplanes $\mathbf{h}_1$ and $\mathbf{h}_2$ with normal vectors $\mathbf{w}_1, \mathbf{w}_2$ creating four buckets. Data-points are shown as coloured shapes, with similar data-points having the same colour and shape. The hashcode for each data-point is found by taking the dot-product of the feature representation onto the normal vectors ($\mathbf{w}_1$, $\mathbf{w}_2$) of each hyperplane. The resulting projected dimensions are binarised by thresholding at zero (Figure (b)) with two thresholds t_1, t_2. Concatenating the resulting bits yields a 2-bit hashcode for each data-point (indicated by the unfilled squares). For example the projection of data-point a is greater than threshold t_1 and so a '1' is appended to its hashcode. Data-point a's projection onto normal vector $\mathbf{w}_2$ is also greater than t_2 and so a '1' is further appended to its hashcode. The hashcode for data-point a is therefore '11' which is also the label for the top-right region of the feature space in Figure (a).

the hyperplane normal vectors positioned either randomly or in data-aware positions in the feature space. The hyperplanes should ideally partition the space in a manner that gives a higher likelihood that similar data points will fall within the same region, and therefore assigned the same hashcode. In the second step the real-valued projections are quantised into binary ('0' or '1') by thresholding the corresponding projected dimensions[5] typically with a single threshold placed at zero for mean centered data.

Despite the success and wide application of algorithms for hashing-based ANN search there still remains considerable downsides to the manner in which the projection and quantisation steps are typically performed. Locality Sensitive Hashing (LSH), one of the arguably most well-known and widely applied methods for hashing-based ANN search, sets the hashing hyperplanes and the quantisation thresholds in a manner that is *independent* of the distribution of the data. For example, in the variant of LSH for preserving the cosine similarity, the normal vectors of the hashing hyperplanes are randomly sampled from a zero mean unit variance multidimensional Gaussian distribution. This data-oblivious mechanism for generating the hashing hypersurfaces runs a high risk of separating dense areas of the feature space and therefore partitioning related data-points into different hashtable buckets (e.g. points *a* and *b* in Figure 1.3). To ameliorate this low recall problem a typical LSH deployment involves partitioning the data with multiple independent hashtables and presenting the union of all the data-points in the colliding hashtable buckets as candidate nearest neighbours. Unfortunately, the greater the number of hashtables the higher the memory requirements needed for an LSH deployment. The quantisation thresholds are also set in a data-independent manner, typically by thresholding at zero along a projected dimension. In this context a projected dimension is formed from collating the projections from all data-points onto the normal vector to a hyperplane. Unfortunately, the region around zero on a projected dimension is usually the area of highest point density which means that there is a high chance of related data-points falling on opposite sides of the threshold and therefore being assigned different bits.

There is clearly a wide scope for improving the retrieval effectiveness of LSH and many other influential but data-oblivious algorithms for hashing-based approximate nearest neighbour search by tackling both of these issues head on: it is this task that forms the focus of this thesis. Specifically I am interested in maximising the *neighbourhood preservation* of both of these steps in the pipeline, that is the preservation of

[5] I define a projected dimension as the collection of the real-valued projections (dot products) of all data-points onto the normal vector to a hyperplane.

the relative distances in the input feature space in the corresponding Hamming space, as this will directly translate into compact binary codes that are more similar for similar data points, yielding a corresponding increase in retrieval effectiveness. Furthermore, there is the added demand that this criterion be met with the shortest possible length of hashcode to conserve storage space and computation time.

1.2 Thesis Contributions

In this thesis I undertake a detailed study of the projection and quantisation operations as they relate to hashing-based ANN search. As identified in Section 1.1, both steps are crucial components in the process which many existing hashing-based ANN search models use to generate similarity preserving hashcodes for data-points. The central hypothesis examined in this thesis can be compactly stated as follows:

The retrieval effectiveness of existing hashing-based ANN search methods can be significantly improved by learning the quantisation thresholds and hashing hyperplanes in a manner that is directly influenced by the distribution of the data.

This thesis statement forms the backbone of the dissertation and all contributions and experiments are focused on gathering evidence to demonstrate its validity. Recall from Section 1.1 that a popular algorithm for solving the ANN search problem, Locality Sensitive Hashing (LSH), firstly picks a *random* dimension in the input feature space and then projects a data-point onto this dimension. This projection step is then followed by a quantisation operation that converts the projection into binary (0 or 1) by *thresholding at zero* with a *single threshold*. The projection and quantisation operations are repeated K times to build up a K-bit hashcode for a given data-point. This hashcode generation pipeline effectively relies on the following three underlying assumptions[6]:

- A_1: Single static threshold placed at zero (for mean centered data)
- A_2: Uniform allocation of thresholds across each dimension

[6]These three specific assumptions were chosen as they are the simplest assumptions that are frequently made in the literature, and whose relaxation I thought would result in the largest gain in retrieval effectiveness. This list is not exhaustive and other limiting assumptions exist such as learning the hyperplanes independently of each other. Examination of these assumptions is left to future work (Chapter 8).

- A_3: Linear hypersurfaces (hyperplanes) positioned randomly

These three assumptions permeate the learning to hash literature and can be found, in some form or another, in most existing work in the field (Indyk and Motwani (1998); Weiss et al. (2008); Liu et al. (2012); Gong and Lazebnik (2011); Raginsky and Lazebnik (2009); Kulis and Darrell (2009)). In this dissertation, for each identified assumption, I introduce a novel data-driven algorithm that relaxes that assumption. In each case I evaluate the proposed algorithm with respect to state-of-the-art baselines on standard image retrieval datasets and demonstrate statistically significant increases in retrieval effectiveness. There are three advantages to the algorithms presented in this thesis: firstly, they can be used to improve the retrieval effectiveness of almost *any* existing model for hashing-based ANN search not just LSH. Secondly a simple extension permits cross-domain applicability, such as retrieving images using a textual query. Thirdly, as a further contribution, I will show in Chapter 7 that the algorithms can be *combined* in a synergistic manner to increase the retrieval effectiveness over what is possible using either algorithm in isolation, at the expense of additional training time.

The specific contributions of this thesis to relaxing these three assumptions are:

A_1) **Single static threshold placed at zero (for mean centered data)**: I address assumption A_1 by formulating a new quantisation algorithm that assigns *multiple thresholds* for each projected dimension, rather than a single threshold. I show that retrieval effectiveness depends heavily on the correct positioning of the threshold(s) along a projected dimension and that a static positioning is sub-optimal. To learn the optimal threshold positions I introduce a novel semi-supervised clustering algorithm that directly optimises an F_1-measure based objective function computed from a data-point adjacency matrix. This objective function seeks to maximise the number of true nearest neighbours assigned identical bits while minimising the number of unrelated data-points receiving the same bits. Under the same bit budget constraint I demonstrate an improved retrieval effectiveness from a multi-threshold quantisation versus a single threshold quantisation. This work is presented in Chapter 4.

A_2) **Uniform allocation of thresholds across each dimension**: It is usually the case that a collection of hyperplanes are not equally informative about the structure of the input feature space. For example, hyperplanes generated by solving an eigenvalue problem tend to capture the majority of the variance of the data in a small subset of the eigenvectors with the largest associated eigenvalues (Gong and Lazebnik (2011)). This means that the lower-variance hyperplanes are unreliable and typically do not capture any meaningful structure in the input space. Intuitively we would like to maximally exploit the additional structure captured by the more informative hyperplanes so as to generate more discriminative hashcodes. I show that this is possible by learning a *variable threshold allocation* across hyperplanes. I relax assumption A_2 and allocate more thresholds (bits) from a fixed threshold budget to the more informative hyperplanes and less thresholds (bits) to the less informative hyperplanes. I propose two novel algorithms for computing a non-uniform allocation of thresholds across projected dimensions. For certain classes of projection functions both algorithms demonstrate significantly improved effectiveness over a uniform threshold allocation. This work is presented in Chapter 5.

A_3) **Linear hypersurfaces placed at random in the feature space**: I introduce a novel three-step iterative hashing algorithm that *learns linear or non-linear hypersurfaces* based on the distribution of the data. Hashcodes initialised from an existing hashing scheme such as LSH, are regularised in step A over an adjacency matrix derived from the training dataset. This step has the effect of setting the hashcode for a given data-point to be the average of the hashcodes of its nearest neighbours (as specified by the adjacency matrix). In the second step (B) a set of binary classifiers are learnt to predict the regularised hashcodes with maximum margin. The regularised hashcodes are then re-labelled using the learned classifiers in Step C and these re-labelled bits are then fed into Step A, with the three steps repeating for a fixed number of iterations. I report significant increases in retrieval effectiveness for hashing with hyperplanes that are specifically adapted to the distribution of the data. I obtain a further boost in retrieval effectiveness through learning non-linear hypersurfaces induced by a radial-basis function (RBF) kernel. This work is presented in Chapter 6.

This hypersurface learning algorithm is then extended to the *cross-modal* hashing scenario in which the query and database points are now in *two* different feature spaces (e.g. text and image descriptors). This extension requires a straightforward change to the unimodal hypersurface learning algorithm: rather than learning K hypersurfaces in a single feature space, I now learn $2K$ hypersurfaces, one set of K hypersurfaces in each of the two feature spaces. Within the textual modality the algorithm is identical to the unimodal model: the annotation hashcode bits are regularised over the data affinity graph and K hypersurfaces are learnt using the textual features and the textual hashcode bits as labels. To form the cross-modal bridge I simply use the regularised hashcode bits in the textual modality as the labels to position K hypersurfaces in the visual feature space. This latter step encourages the hypersurfaces in the two modalities to form buckets that are consistent in both feature spaces. Experimental evaluation of the multimodal hashing scheme demonstrates state-of-the-art cross-modal retrieval effectiveness in comparison to strong multimodal hashing baselines from the literature.

Finally I bring together all the contributions of this thesis by learning both the hashing hyperplanes *and* quantisation thresholds together in the same model, overcoming the main limitation of previous work where either the projection function or quantisation function, or both, remain uninformed by the data distribution. To achieve this learning objective I combine the multi-threshold quantisation algorithms introduced as a means of relaxing assumptions A_1-A_2 with the projection function introduced to relax assumption A_3. The result is a new hashing model for ANN search that is fully flexible, adapting the positioning of the hyperplanes and the quantisation thresholds to the statistics of a given dataset. In the experimental evaluation it is shown conclusively that this combination of models exhibits a retrieval effectiveness greater than using either component in isolation. This result neatly unifies the main contributions of this thesis while also revealing a potentially fruitful new research direction in which the entire hashing model becomes data-adaptive. This work is presented in Chapter 7.

1.3 Thesis Outline

The remainder of this document is structured as follows:

Chapter 2: Background, provides background on the problem of hashing-based ANN search and a review of existing relevant research, placing my contributions in the context of prior-art. The learning to hash research field is at a point where a consolidated review of previous existing work is required. I therefore contribute one of the first thorough reviews of the field encompassing the important functions of quantisation and projection.

Chapter 3: Experimental Methodology, introduces the standard datasets, evaluation paradigms and evaluation metrics used in the literature. I also identify and suggest corrections to certain flaws in the way existing work is evaluated.

Chapter 4: Learning Multiple Quantisation Thresholds, outlines my quantisation

algorithm for positioning multiple thresholds along each projected dimension. I show how the model optimises threshold positions based on maximisation of an F_β-measure objective computed on a data-point adjacency matrix. This objective function encourages a positioning of the thresholds so that more related data-points fall within the same quantised regions and thereby are assigned similar bits. I describe how a brute-force optimisation of the threshold positions is computationally intractable and introduce an efficient stochastic search algorithm that rapidly finds a good local optima in the F_β-measure objective function.

Chapter 5: Learning Variable Quantisation Thresholds, relaxes the uniform threshold allocation assumption introduced by the quantisation model presented in Chapter 4. Specifically I examine the benefits of *varying* the number of thresholds allocated per projected dimension. The two proposed adaptive threshold learning algorithms are found to be more effective for image retrieval than the model presented in Chapter 4.

Chapter 6: Learning the Hashing Hypersurfaces, departs from Chapters 4-5 and focuses on the complementary problem of data-dependent projection function learning. I show how this problem reduces to the optimisation of a set of hypersurfaces in the input feature space guided by must-link and cannot-link constraints on the data-points pairs. I introduce a novel three-step iterative algorithm for learning hashing hyperplanes and show that it can be readily extended to learn non-linear hypersurfaces induced by the radial basis function kernel. I discuss how the model exhibits a number of considerable advantages over previous work, most notably the absence of a computationally expensive matrix factorisation. I then further extend the model to tackle the task of cross-modal retrieval where the query and database data-points are in two different feature spaces (for example, image and textual descriptors).

Chapter 7: Learning Hypersurfaces and Quantisation Thresholds, provides a preliminary exploration into the effects of combining multiple complementary relaxations in the same hashing model. The multi-threshold optimisation algorithms introduced in Chapters 4-5 are used to quantise the projections resulting from the hypersurface learning algorithm introduced in Chapter 6. This chapter unifies my main contributions to hypersurface and quantisation threshold learning in a single model for hashing-based ANN search. I show the important result that relaxing multiple assumptions as part of the same model can have an additive benefit on retrieval effectiveness.

Chapter 8: Conclusions and Future Work, summarises the main contributions in

this thesis and reviews the central claim set out in Chapter 1 in the context of the results presented in Chapters 4-7. Potential fruitful avenues for future research are also proposed.

1.4 Published Work

Chapter 4 expands on the work previously published in SIGIR'13 by providing more detail on the stochastic search algorithms used for threshold learning and giving additional experimental results and analysis.

- Moran, S. and Lavrenko, V. and Osborne, M. (2013). Neighbourhood Preserving Quantisation for LSH. In *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. Dublin, Ireland.

Chapter 5 provides more detail on the work previously published in ACL'13. Specifically, I introduce a second greedy algorithm for solving the variable threshold allocation problem and provide an expanded set of experiments.

- Moran, S. and Lavrenko, V. and Osborne, M. (2013). Variable Bit Quantisation for LSH. In *Association for Computational Linguistics (ACL)*. Sofia, Bulgaria.

The unimodal part of Chapter 6 was previously published as follows:

- Moran, S. and Lavrenko, V. (2015). Graph Regularised Hashing. In *European Conference on Information Retrieval (ECIR)*. Vienna, Austria.

The cross-modal part of Chapter 6 was previously published in SIGIR'15:

- Moran, S. and Lavrenko (2015). Regularised Cross Modal Hashing. In *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. Santiago, Chile.

Chapter 7 was previously published in SIGIR'16:

- Moran, S. (2016). Learning to Project and Binarise for Hashing Based Approximate Nearest Neighbour Search. In *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. Pisa, Italy.

Chapter 2

Background

2.1 Introduction

In this chapter I provide an overview of existing research that is crucial for understanding the contributions made in this thesis. The chapter begins in Section 2.3 with an introduction to nearest neighbour (NN) search, why the problem is important and how it can be solved. This introduction is then followed by a discussion in Section 2.3 as to why a relaxed version of the problem is required, known commonly as approximate nearest neighbour (ANN) search. In Section 2.4, I describe a seminal method, Locality Sensitive Hashing (LSH), for solving the ANN search problem in a time constant in the number of data-points. The limitations of LSH are discussed and I use those drawbacks as a motivation for a review of a host of more recently proposed algorithms for ANN search that demonstrate a higher retrieval effectiveness on the task of image retrieval. I divide this latter part of the review into methods for binary quantisation (Section 2.5) and projection function learning (Section 2.6), mirroring the two stages of hashcode generation first introduced in Chapter 1.

2.2 Preliminaries and Notation Definition

This thesis adheres to the standard typography for vectors $\mathbf{x}$ (lowercase bold) and matrices $\mathbf{X}$ (uppercase bold). The ijth entry of matrix $\mathbf{X}$ is denoted by an uppercase, non-bold letter X_{ij}. Vectors $\mathbf{x} = [x_1, x_2 \ldots, x_N]^\intercal$ are assumed to be column vectors formed by stacking N scalar values. $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\intercal$ signifies the stacking of the N column vectors $\left\{\mathbf{x}_i \in \mathbb{R}^D\right\}_{i=1}^{N}$ *row-wise* to form matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$. I use the notation $\mathbf{x}^c = \mathbf{X}_{\bullet c}$ to refer to the vector of elements in the c^{th} column of matrix $\mathbf{X}$. In a similar manner $\mathbf{x}_r = \mathbf{X}_{r\bullet}$ denotes the vector of elements in the r^{th} row of matrix $\mathbf{X}$. Functions are indicated by lowercase, non-bold letters e.g $d(.,.)$. I summarise the notation used throughout this thesis in Table A.1 situated in Appendix A.

The related work described in this chapter all share the same problem definition. We are given a dataset consisting of N points $\mathbf{X} \in \mathbb{R}^{N \times D} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\intercal$ where each data-point point $\mathbf{x}_i \in \mathbb{R}^D$ is a D-dimensional vector of real-valued features. The objective is to construct K hash functions $\{h_k : \mathbb{R}^D \rightarrow \{0,1\}\}_{k=1}^K$ the output of which can be concatenated as $[h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_K(\mathbf{x}_i)]$ to yield a binary embedding function $\left\{g_l : \mathbb{R}^D \rightarrow \{0,1\}^K\right\}$ that maps each data-point $\mathbf{x}_i$ to a K-bit binary hashcode $\mathbf{b}_i \in \{0,1\}^K$. For the embedding functions to be useful for nearest neighbour search we will require the bits to be selected in such a way that similar points $\mathbf{x}_i, \mathbf{x}_j$ will have similar hashcodes $\mathbf{b}_i, \mathbf{b}_j$, as measured by an appropriate distance function in the hashcode space such as the Hamming distance. I dedicate the remainder of this chapter to describing how similarity preserving hash functions are constructed by relevant models from the literature. A birds-eye-view of the structure of this chapter is shown in Figure 2.1.

While the hashing models discussed in this chapter are evaluated solely on image datasets, they are by no means restricted to this particular data-type. A powerful property of the discussed hashing models is their applicability to data whose instances can be represented as vectors of a certain dimensionality, and this includes text and audio data. We may find, however, that the relative performance of the models changes depending on the data-type of interest. For example, high dimensional and sparse textual vectors are expected to cause scaling issues for the eigendecomposition-based models described in this chapter, which work particularly well on the low-dimensional, dense feature vectors found in the field of Computer Vision. An investigation into how the described models perform on datasets of different types (text, audio, vision) would be an interesting avenue for future work.

2.3 Approximate Nearest Neighbour (ANN) Search

In this section I first formally define the problem of nearest neighbour (NN) search which I informally introduced in Chapter 1. I will then examine the relaxed version of NN search known as *approximate* NN search, the field upon which this thesis builds, and describe how it differs from alternative algorithms for solving the NN search problem.

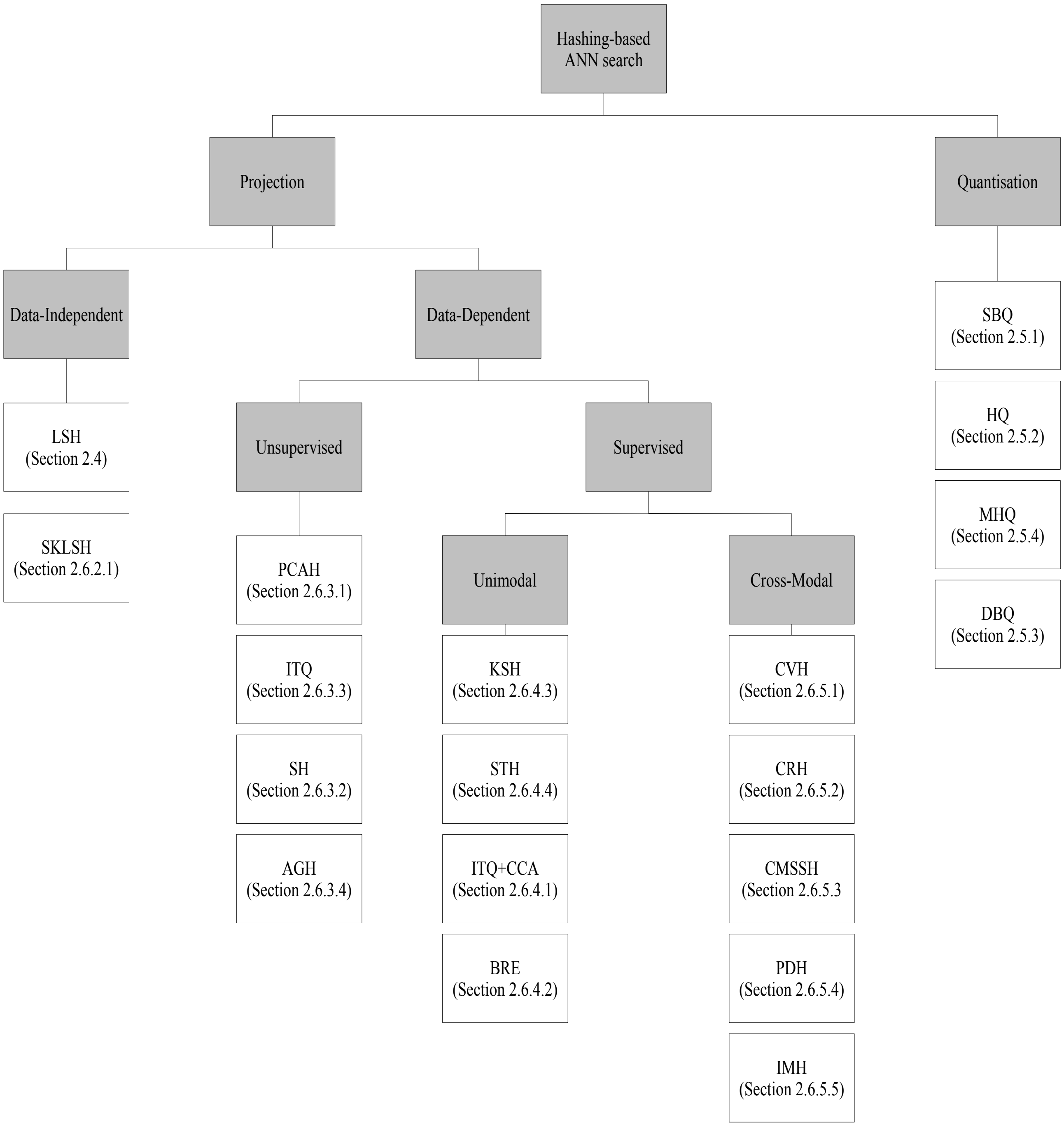


Figure 2.1: Overview of one possible categorisation of the field of hashing-based ANN search. The main categories are shown in the grey boxes while the actual models themselves are highlighted in white alongside their relevant section number in this chapter.

Nearest neighbour search can be defined as the problem of retrieving the closest data-point $NN(\mathbf{q})$ to a query $\mathbf{q} \in \mathbb{R}^D$ in a database of N data-points $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^\intercal$ where $\mathbf{x}_i \in \mathbb{R}^D$. The similarity between data-points is defined by a distance function of interest $\{d(.,.) : \mathbb{R}^D \times \mathbb{R}^D \rightarrow [0,1]\}$. This variant of the problem is also known as 1-NN search and is specified mathematically in Equation 2.1

$$NN(\mathbf{q}) = argmin_{\mathbf{x}_i \in \mathbf{X}} d(\mathbf{x}_i, \mathbf{q}) \tag{2.1}$$

It is straightforward to generalise this problem definition to return the closest K neighbours to the query. This variant is popularly referred to as k-NN search and is a fundamental component in a wide range of different machine learning methods including non-parametric kernel density estimation (Bishop (2006), Ulz and Moran (2013), Moran and Lavrenko (2014)). The distance function $d(.,.)$ between the data-points is typically computed using a generic distance metric such as the l_p-norm (Equation 2.2)

$$\begin{aligned} d_{pnorm}(\mathbf{x}_i, \mathbf{x}_j) &= \quad \|\mathbf{x}_i - \mathbf{x}_j\|_\rho \\ &= \left(\sum_{k=1}^{D} |x_{ik} - x_{jk}|^\rho \right)^{\frac{1}{\rho}} \end{aligned} \tag{2.2}$$

The parameter $\rho \in \mathbb{R}_+$. Setting $\rho = 1$ yields the Manhattan distance and $\rho = 2$ gives the Euclidean distance while $\rho < 1$ introduces the Minkowski family of fractional distances. The cosine distance presented in Equation 2.3 is another popular distance metric for NN search that has proven particularly effective for document retrieval (Manning et al. (2008), Ravichandran et al. (2005))

$$d_{cosine}(\mathbf{x}_i, \mathbf{x}_j) = 1 - \frac{\sum_{k=1}^{D} x_{ik} x_{jk}}{\sqrt{\sum_{k=1}^{D} x_{ik}^2} \sqrt{\sum_{k=1}^{D} x_{jk}^2}} \tag{2.3}$$

We will also come across the Hamming distance extensively in this thesis as it is the de-facto metric for comparing binary strings (Equation 2.4)

$$d_{hamming}(\mathbf{b}_i, \mathbf{b}_j) = \sum_{k=1}^{D} \delta[b_{ik} \neq b_{jk}] \tag{2.4}$$

The function $\delta(.) = 1$ if its argument is true, and 0 otherwise. The Hamming distance therefore counts the number of corresponding dimensions (bits) that are *not equal* in the two hashcodes.

These generic distance metrics do not adapt to the distribution of the data, measuring the distances between data-points in the same way regardless of the specifics of the dataset. In applying both metrics in practice we implicitly hope that the resulting distances correlate well with the specific notion of similarity required for the domain. For example, in the field of image annotation that the Euclidean distance between the feature representation of two images can tease apart an image of a cat from that of a dog. In many cases this is an unrealistic assumption that leads to low retrieval effectiveness (Kulis (2013); Moran and Lavrenko (2014)). Distance metric learning is an active research field dedicated to learning distance metrics tuned to a specific dataset. These methods typically learn a scaling and a rotation of the data so that the Euclidean distance in the transformed space correlates better with, for example, class-based supervision. Perhaps unsurprisingly learnt metrics have been shown to greatly improve the quality of NN retrieval over and above their non data-adaptive counterparts such as the l_p-norm (Kulis (2013)). I pick up this thread again in Section 2.6 where I reveal how this important idea of data-dependent distance functions has inspired recent developments in the field of hashing-based ANN search.

To search for NNs to a query we need to construct a data-structure or algorithm that takes our selected notion of distance and retrieves data-points that are close to the query under that specific distance metric. Brute force search is a straightforward algorithm for solving the nearest neighbour search problem with any desired distance metric. In brute-force search the distance to every data-point in the database is computed and the data-point(s) with the smallest distance to the query returned as the nearest neighbour(s). The advantages of brute force search are its simplicity of implementation and its guarantee that the closest nearest neighbours will eventually be retrieved. However, exhaustively comparing the query to every data-point in the database gives a linear $O(ND)$ time complexity which quickly makes brute force search intractable for nearest neighbour search across datasets with many data-points (N) and a moderate to high dimensionality (D). In this situation a more informed approach to the nearest neighbour search problem is required.

The generality and importance of nearest neighbour search, described in detail in the motivation for this thesis in Chapter 1, ensures that the problem remains an active research area within many scientific disciplines including Information Retrieval (IR) and Computer Vision. Efficient multidimensional indexing data-structures for NN search have been proposed for data-points of low-dimensionality (usually $D \leq 10$), with some of the more well known examples of this kind being the KD-tree (Bentley

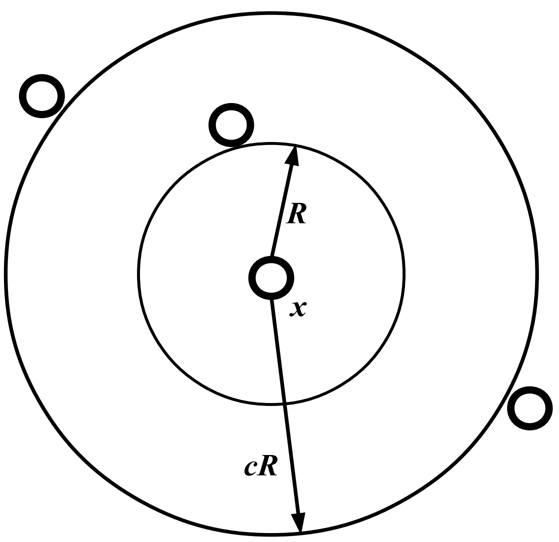


Figure 2.2: The (c,R)-approximate NN problem: in many applications it is acceptable to return a data-point (indicated with the circles) within a distance cR of the query point $\mathbf{x}$, where R is the distance to the exact NN and the approximation factor $c > 1$.

(1975)), quad-tree (Finkel and Bentley (1974)), X-tree (Berchtold et al. (1997)) and SR-tree (Katayama and Satoh (1997)). Unfortunately it has been shown that methods relying on a space partitioning or clustering of the input feature space can do no better than brute-force search in high dimensions (Weber et al. (1998)). This result severely limits the applicability of these algorithms to image and document collections where it is not uncommon to find feature representations with hundreds, thousands or indeed millions of dimensions. The impossibility of retrieving *exact* nearest neighbours in sub-linear time in high dimensional spaces is one particular incarnation of the well-known *"curse of dimensionality"* (Minsky and Papert (1969)).

Algorithms for *approximate* nearest neighbour search circumvent the curse of dimensionality by relaxing the need for an optimal (exact) solution to the problem, in return for a substantially improved bound on the query time. In many practical scenarios, for example detecting a large number of object classes (Dean et al. (2013)) or matching variable sized sets of features (Grauman and Darrell (2007)), retrieving sub-optimal nearest neighbours is an entirely acceptable compromise for a greatly reduced query time. In the theoretical computer science literature the problem is commonly referred to as the (c,R)-approximate NN decision problem. In this problem definition, we are happy to accept a nearest neighbour within distance cR of the query, where c is an approximation factor ($c > 1$) and R is the distance to the exact NN (Figure 2.2). The (c,R)-approximate NN decision problem is formalised in Andoni and Indyk (2008);

Petrovic (2012) and defined in Definition 2.3.1:

Definition 2.3.1. Randomised c-approximate R-near neighbour problem: given a set of N data-points in a D dimensional space, return each cR-nearest neighbour of the query data-point $\mathbf{q}$ with probability 1-δ, where $\delta > 0, R > 0$.

The approximation factor c effectively determines the degree of sub-optimality in the returned nearest neighbours that we are willing to tolerate. The greater the value of c the more distant the returned nearest neighbour might be from the optimal nearest neighbour, with the advantage of a reduction in the query time. This clear trade-off between effectiveness and efficiency lies at the heart of effective algorithms for solving the (c,R)-approximate nearest neighbour problem.

The R-near neighbour reporting problem (Andoni and Indyk (2008); Petrovic (2012)) is similar but without the approximation factor c (Definition 2.3.2)

Definition 2.3.2. Randomised R-near neighbour problem: given a set of N data-points in a D dimensional space, return each R-nearest neighbour of the query data-point $\mathbf{q}$ with probability 1-δ, where $\delta > 0, R > 0$

In Section 2.4, I will introduce Locality Sensitive Hashing (LSH), a family of algorithms that provide a concrete method for solving these approximate nearest neighbour search decision problems in constant time per query.

2.4 Locality Sensitive Hashing (LSH)

The objective for any successful model for hashing-based ANN search is to pre-process the database $\mathbf{X} \in \mathbb{R}^{N \times D}$ so that at query-time nearest neighbours can be found more efficiently than a simple brute force search over the entire database. In this section I introduce the core ideas behind Locality Sensitive Hashing (LSH) (Indyk and Motwani (1998)), one of the most influential algorithms for ANN search and the first to provide a sub-linear time solution to the randomised c-approximate R-near neighbour problem. LSH has found wide-application in vision problems, from recognising 100,000 object classes on a single machine (Dean et al. (2013)), to pose estimation (Shakhnarovich et al. (2003)), bag-of-words indexing (Chum et al. (2008)) and shape matching (Grauman and Darrell (2004)). The hashing models I introduce later in this literature review (Sections 2.5-2.6) can all be thought of as extensions of LSH that try and overcome certain disadvantages with the original algorithm. Given the central importance of LSH, I therefore spend a considerable proportion of this review in discussing the algorithm in detail. I will begin by giving a general overview of LSH, independent of the similarity metric of interest. In Section 2.4.1 I will then discuss a concrete instantiation of LSH for the inner product similarity that will be of central importance in this thesis.

The key idea behind LSH is to pre-process the database by assigning hashcodes to each data-point in such a way that data-points that are closer in $\mathbb{R}^D$ under some distance metric $\{d(.,.) : \mathbb{R}^D \times \mathbb{R}^D \to [0,1]\}$ have a higher probability of colliding in the same hashtable bucket than data-points that are much further apart in $\mathbb{R}^D$. LSH therefore transforms nearest neighbour search into the process of examining the contents of a small set of hashtable buckets, which is likely to be many times more efficient than exhaustive brute force search over every data-point. The question then arises as to how LSH generates hashcodes (i.e. the indices into the hashtable buckets) which are the same for data-points that are "close" in the original feature-space. To achieve this property, LSH uses what is known as locality sensitive hash functions $\{h_k : \mathbb{R}^D \to U\}$ that map $\mathbb{R}^D$ to some universe U (for example, binary bits or positive integers). The locality sensitive hash functions are drawn uniformly at random from a hash function family $\mathcal{H}$ (Definition 2.4.1)

Definition 2.4.1. Locality sensitive hash function family: a hash function family $\mathcal{H}$ is deemed (R, cR, P_1, P_2) sensitive if for any two data-points $\mathbf{p}$, $\mathbf{q} \in \mathbb{R}^D$:

$$\begin{aligned} &if \quad d(\mathbf{p},\mathbf{q}) \le R \quad then \quad Pr_{\mathcal{H}}(h(\mathbf{p}) = h(\mathbf{q})) \ge P_1 \\ &if \quad d(\mathbf{p},\mathbf{q}) \ge cR \quad then \quad Pr_{\mathcal{H}}(h(\mathbf{p}) = h(\mathbf{q})) \le P_2 \end{aligned}$$

where $Pr_{\mathcal{H}}(h(\mathbf{p}) = h(\mathbf{q}))$ refers to the probability that two data-points hash to the same value given a hash function $h(.)$ chosen uniformly at random from $\mathcal{H}$. If a locality sensitive hash function family is to be useful for nearest neighbour search then we require $P_1 > P_2$ and $c > 1$. In other words there should be a *high* probability P_1 of two data-points $\mathbf{p} \in \mathbb{R}^D, \mathbf{q} \in \mathbb{R}^D$ close by to each other (i.e. $d(\mathbf{p},\mathbf{q}) \le R$) in $\mathbb{R}^D$ colliding in the same hashtable bucket. Conversely there should be a *low* probability P_2 of more distant data-points (i.e. $d(\mathbf{p},\mathbf{q}) \ge cR$) colliding in the same hashtable bucket. In this way the output of a hash function $h(.)$ chosen uniformly at random from $\mathcal{H}$ is intimately tied to the spatial arrangement of the data-points in $\mathbf{X}$ as measured under a distance metric of interest $\{d(.,.) : \mathbb{R}^D \times \mathbb{R}^D \to [0,1]\}$. Ideally we would like that $P_1 = 1$ and $P_2 = 0$ so that all data-points that are within $d(\mathbf{p},\mathbf{q}) \le R$ of the query map to the same hashtable bucket and all data-points with distance $d(\mathbf{p},\mathbf{q}) \ge cR$ map to a different hashtable bucket. Note, the case $R < d(\mathbf{p},\mathbf{q}) < cR$ remains unaddressed, but nevertheless R and cR can be made close at the expense of making P_1 and P_2 undesirably close (Rajaraman and Ullman (2011)).

Fortunately it is possible to construct a wide variety of useful hash function families that have the property that $P_1 > P_2$ and $c > 1$. For example, locality sensitive hash function families have so far been introduced for many distance functions of prime interest such as the L_p distance in $\mathbb{R}^D$ for $p \in [0, 2)$ (Datar et al. (2004)), cosine distance (inner product similarity) (Charikar (2002)), Jaccard distance (Broder (1997)) and L_2-norm on the unit hypersphere (Terasawa and Tanaka (2007)). Choosing the locality sensitive hash function family $\mathcal{H}$ is an important decision that needs to be considered when implementing an LSH system in practice. In a similar way that selecting an appropriate distance function for brute force search is application dependent, so too is choosing a locality sensitive hash function family. For example, the hash function family for the *inner product similarity*, which draws its hash functions uniformly from a unit sphere, has proven to be successful for detecting new events in high-volume document streams (Petrovic (2012), Osborne et al. (2014)). I will expand on this particular hash function family in more detail in Section 2.4.1. The LSH hash function family for the Euclidean distance (Datar et al. (2004)), which randomly samples hash functions from a D dimensional zero mean unit variance Gaussian distribution, has also found wide applicability in applications such as pose estimation (Shakhnarovich et al. (2006); Matei et al. (2006)). This hash function family relies on the *Johnson-Lindenstrauss* lemma (Johnson and Lindenstrauss (1984)) as a guarantee that there will be limited distortion to the pairwise distances in the lower-dimensional embedding space.

The usefulness of any locality sensitive hash function family for nearest neighbour search is dependent on the *gap* between P_1 and P_2 which dictates the collision probabilities between points in the range $[0, R]$ in which the R-near neighbours are to be found and (cR, ∞). If the gap between P_1 and P_2 is small then a query will have a similar probability of mapping to the hashtable bucket of a distant data-point as it will be to a close-by data-point. Without a sufficient difference between P_1 and P_2 the quality of nearest neighbour search under LSH will be poor with a high number of false positives and false negatives. The gap between P_1 and P_2 can be *amplified* by concatenating together K randomly selected hash functions to create an embedding function into a K dimensional space. This multidimensional embedding function is given by Equation 2.5

$$g_l(\mathbf{q}) = [h_1(\mathbf{q}), h_2(\mathbf{q}), \dots, h_K(\mathbf{q})] \tag{2.5}$$

where $g_l(.)$ is drawn uniformly at random from function family $\mathcal{G} = \left\{\mathbb{R}^D \rightarrow U^K\right\}$ and $h_k \in \mathcal{H}$. This concatenation of K hash functions increases the gap between P_1 and P_2, amplifying the difference between the probabilities of collisions between nearby and far data-points. For an embedding function $g_l(.)$ the probability $P(g_l(\mathbf{q}) = g_l(\mathbf{p}))$ that the hashcodes for any two distant data-points $\mathbf{p} \in \mathbb{R}^D, \mathbf{q} \in \mathbb{R}^D$ with $d(\mathbf{p},\mathbf{q}) \geq cR$ will match is given by $P_2^{'} = P_2^K$. This reduction in the number of false positives with increasing K is the underlying motivation for using multiple bits in a hashcode: as K increases there is a gradually lower probability that distant data-points will collide in the same hashtable bucket as the query $\mathbf{q}$. However, increasing K also reduces the probability of collision between nearby data-points (by $P_1^{'} = P_1^K$), and so while the precision increases through elimination of false positives we will also suffer a decrease in recall due to the introduction of false negatives. For a judicious choice of K, if $P_1 > P_2$, it is possible to keep probability $P_1^{'}$ bounded significantly away from zero, while moving probability $P_2^{'}$ close to zero (Rajaraman and Ullman (2011)).

In practice for a large enough hashcode length K, we might find that very few close by data-points ($d(\mathbf{p},\mathbf{q}) < R$) collide in the same hashtable bucket as no data-points are likely to share an identical hashcode. The number of buckets in a single hashtable grows at an exponential rate (2^K) as the hashcode length is increased and so many of these buckets will be empty for a large enough setting of K. The other LSH parameter is the number L of embedding functions sampled from $\mathcal{G}$, with each embedding function indexing into one of L independent hashtables. The value of L can be increased to counteract the lower level of recall that arises from a longer hashcode length K. The probability that two hashcodes will collide in the same hashtable bucket for *at least one* hashtable is then given by the expression $P_1^{''} = (1-(1-P_1^K)^L)$. Even though using multiple hashtables will increase probabilities $P_1^{''}$ and $P_2^{''}$, it is possible to set L so as to increase probability $P_1^{''}$ towards one, while also keeping $P_2^{''}$ bounded significantly away from one (Rajaraman and Ullman (2011)). Therefore, the parameters K and L can be set in combination so as to cause probability $P_1^{''}$ to be close to one, while moving $P_2^{''}$ close to zero, which is the property we seek for an ideal locality sensitive hash function (an illustration of this effect for various settings of K and L is shown in Figures 2.3-2.4). Of course, the higher the values of K and L the greater the computation time required for the actual hashing.

The setting of L and K permits the practitioner trade-off of the precision and recall achieved while choosing an appropriate overall computational cost. One possible strategy for setting L and K is to use the probabilistic bounds on the failure probability

Algorithm 1: LSH PRE-PROCESSING STEP (PETROVIC (2012))

```
Input: Data-points $\mathbf{X} \in \mathbb{R}^{N \times D}$, $L$ embedding functions $[g_1(.), \ldots, g_L(.)]$ with
        $g_l(.) = \{h_{l1}(.), \ldots, h_{lK}(.)\}$, $h_{lk}(.)$ selected uniformly from family $\mathcal{H}$
Output: Data-points indexed into the buckets of $L$ hashtables $H$
1 for $i \leftarrow 1$ to $L$ do
2     for $j \leftarrow 1$ to $|\mathbf{X}|$ do
3         Insert $\mathbf{x}_j$ into bucket $H[i][g_i(\mathbf{x}_j)]$
4     end
5 end
6 return $H$
```

offered by LSH. The setting of L and K can be found by firstly deciding on an acceptable probability $\delta < (1 - P_1^K)^L$ of LSH failing to find an R-nearest neighbour with a specified similarity (P_1) to the query. The setting of L guaranteeing the failure probability δ for a given hashcode length K is then given by $L \geq \lceil log(\delta)/log(1 - P_1^K) \rceil$. The E^{2}LSH[1] package recommends choosing the hashcode length K to minimize the mean query time for all data-points a dataset. Some LSH implementations attempt to eliminate the need to choose these parameters altogether, see for example LSH-forest (Bawa et al. (2005)). For the practitioner, Petrovic (2012) provide an enlightening discussion on how the best fitting L and K parameters were chosen for an LSH-based event detection system. This system was successfully used for real-time detection, tracking, and monitoring of automatically discovered events in social media streams (Osborne et al. (2014)).

Having chosen the desired hash function family $\mathcal{H}$ and the setting of K and L there are two final steps to using LSH for nearest neighbour search: *pre-processing*, in which the database points are hashed using the L multidimensional embedding functions $\{g_l : \mathbb{R}^D \rightarrow \{0,1\}\}_{l=1}^{L}$ into the buckets of L hashtables $g_l(\mathbf{p})$ for $l = \{1 \ldots L\}$; and *querying*, where the query is also hashed using the same hash functions and the nearest neighbours retrieved from the colliding hashtable buckets $\{g_1(\mathbf{q}), \ldots, g_l(\mathbf{q})\}$. Typically, the distance (e.g. Euclidean or cosine) from the query to each of the data-points in this candidate list of nearest neighbours is then computed and any data-points $> R$ discarded. The pre-processing step is presented in Algorithm 1 while the querying process is presented in Algorithm 2. The presentation of the pre-processing and

[1] `http://www.mit.edu/~andoni/LSH/`

Algorithm 2: LSH QUERYING STEP (PETROVIC (2012))

```
Input: Query $\mathbf{q} \in \mathbb{R}^D$, Database $\mathbf{X} \in \mathbb{R}^{N \times D}$, $L$ functions $[g_1(.), \ldots, g_L(.)]$ with
       $g_l = \{h_{l1}(.), \ldots, h_{lK}(.)\}$ and $h_k$ selected uniformly at random from a
       hash function family $\mathcal{H}$, hashtables $H$
Output: The set $S$ of $R$ (strategy 2) nearest neighbours of $\mathbf{q}$
1 for $i \leftarrow 1$ to $L$ do
2     for $j \leftarrow 1$ to $|H[i][g_i(\mathbf{q})]|$ do
3         Retrieve next data-point $\mathbf{x}_j$ from bucket $H[i][g_i(\mathbf{q})]$
4         if $(d(\mathbf{x}_j, \mathbf{q}) < R)$ then          // Query strategy 2
5             Put $\mathbf{x}_j$ into retrieved set $S$
6         end
7     end
8 end
9 return $S$
```

querying algorithms has been inspired by a similar specification in Petrovic (2012). There are two LSH querying strategies and both are directly related to the two variants of the approximate nearest neighbour decision problem presented in Definitions 2.3.1-2.3.2. The strategy presented in Algorithm 2 solves the R-near neighbour reporting problem (Definition 2.3.2) as all data-points in the colliding hashtable buckets are examined. In the unlikely worst case scenario this latter strategy may cause the search to examine every data-point in the database. The randomised c-approximate R-near neighbour problem (Definition 2.3.1) is solved by stopping the search after the first $L' = 3L$ data-points have been retrieved. This strategy comes with an $O(L)$ bound on the query time.

2.4.1 LSH with Sign Random Projections

In this dissertation I will be primarily interested in the locality sensitive hash function family for the *inner product similarity* which traditionally has been used as a baseline for comparison by existing research in the learning to hash literature. The inner product similarity is defined in Equation 2.6.

$$\begin{aligned} d(\mathbf{p},\mathbf{q}) &= \sum_{k=1}^{D} p_k q_k \\ &= \mathbf{p}^T\mathbf{q} \end{aligned} \tag{2.6}$$

Equation 2.6 can also be interpreted as the cosine similarity between two L_2 *normalised* vectors mapped on the unit sphere. The cosine similarity measures the closeness between two data-points based on the angle (θ) between their respective vectorial representations in the D-dimensional space, which could be a GIST descriptor (Oliva and Torralba (2001)) for an image or a TF-IDF vector for a document. As the angle between the two vectors widens their cosine similarity decreases, and vice-versa. The locality sensitive hash function family for the cosine similarity $\mathcal{H}_{cosine}$ is formulated by intersecting the space with K hyperplanes drawn randomly from a multidimensional Gaussian distribution with mean zero and unit variance. Depending on what side of a hyperplane a data-point falls, its hashcode is appended with either a '0' or a '1'. The intuition is as follows: the greater the angle between a query $\mathbf{q} \in \mathbb{R}^D$ and a database point $\mathbf{p} \in \mathbb{R}^D$ the more probable it is that the space between the vectors will be partitioned by a randomly drawn hyperplane. The greater the angle, the more often the intervening space will be partitioned by random hyperplanes and the lower the number of bits the hashcodes will share in common. We have achieved the desired property: the output of a hash function (randomly drawn hyperplane) is less likely to match as the angle between two data-points is increased.

More formally, a randomly drawn hash function h_k from $\mathcal{H}_{cosine}$ has the specification given in Equation 2.7

$$\begin{aligned} h_k(\mathbf{q}) &= \frac{1}{2}(1 + sgn(\mathbf{w}_k^{\intercal}\mathbf{q})) \\ &= q_k(p_k(\mathbf{q})) \end{aligned} \tag{2.7}$$

where sgn is the sign function adjusted so that $sgn(0) = -1$, $\mathbf{w}_k \in \mathbb{R}^D$ denotes the normal vector of hyperplane $\mathbf{h}_k$. I denote as $\left\{p_k : \mathbb{R}^D \rightarrow \mathbb{R}\right\}$ the *projection function* (a dot product in this case) that maps a data-point onto a randomly chosen dimension. Here $\left\{q_k : \mathbb{R} \rightarrow \{0,1\}^B\right\}$ denotes the *quantisation function* (thresholding at zero with the sign function in this case) that converts the projection of a data-point into one or more binary bits. Equation 2.7 is the mathematical procedure for determining the position of a data-point with respect to a separating hyperplane, with a '0' or a '1' output depending on the side. With the hash function as specified in Equation 2.7, Goemans and Williamson (1995) showed that the probability of a collision is given as

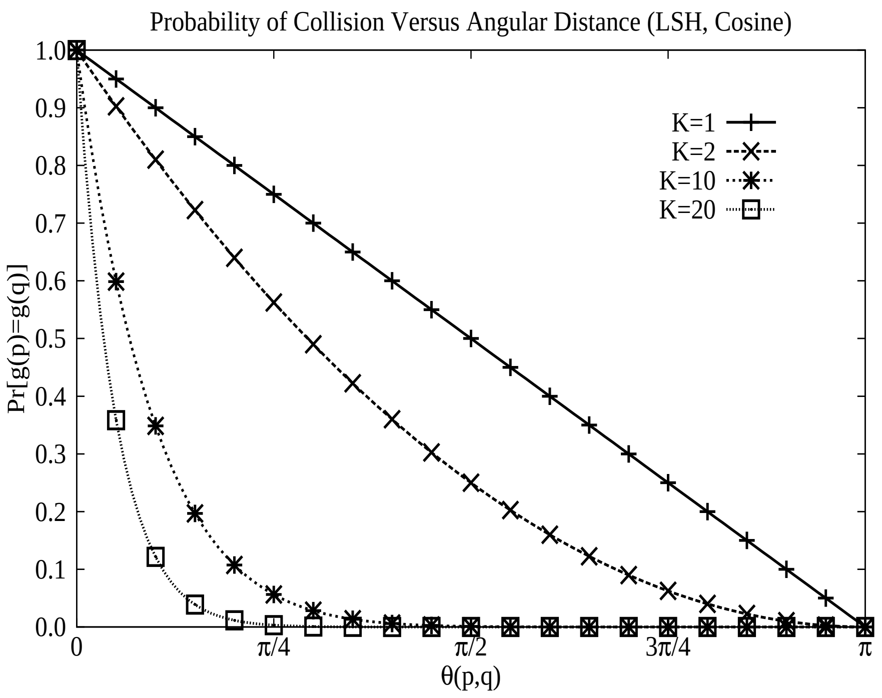


Figure 2.3: Visualising the probability of two hashcodes $g(\mathbf{p}), g(\mathbf{q})$ matching against the length of the hashcode key ($K = 1, 2, 10, 20$). The hash function family is $\mathcal{H}_{cosine}$. As the hashcode length becomes longer the two data-points must be close together (in terms of angle) in order for their collision probability to be high. This is intuitive because if we draw more hyperplanes (bits) there is a greater chance of the two data-points falling on different sides of at least one of the hyperplanes, particularly if the data-points are spaced further apart. This figure is adapted from a similar chart in Petrovic (2012).

in Equation 2.8.

$$Pr_{\mathcal{H}_{cosine}}(h(\mathbf{p}) = h(\mathbf{q})) = 1 - \frac{\theta(\mathbf{p},\mathbf{q})}{\pi} \tag{2.8}$$

Equation 2.8 operationalises our earlier intuition of there being a lower collision probability with a greater angle θ (in radians) when applying a hash function of the form given in Equation 2.7. $\mathcal{H}_{cosine}$ is therefore a $(R, cR, 1 - R/\pi, 1 - cR/\pi)$-sensitive hash function family, where the angular distance R is measured in radians. The amplification strategy I discussed in Section 2.4 for increasing the P_1, P_2 gap works equally as well for $\mathcal{H}_{cosine}$. As before, choosing the length of K with an appropriate setting of the number of hashtables L is crucial for retrieval effectiveness and efficiency in an end-application. I illustrate the locality sensitive nature of $\mathcal{H}_{cosine}$ in Figures 2.3-2.4. In these figures I change the value of K and L and observe the effect on the probability of a collision occurring in the hashtables.

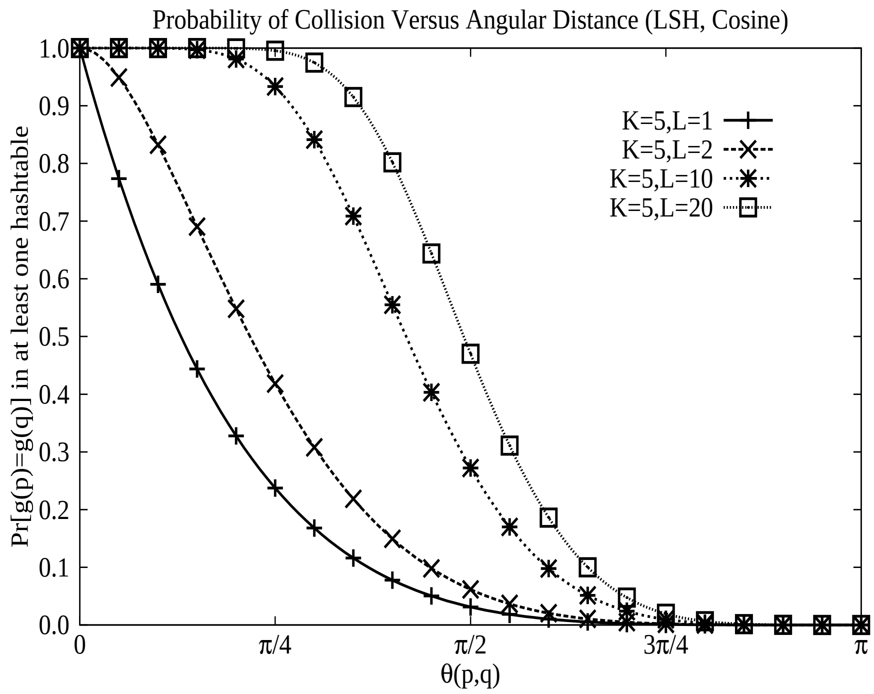


Figure 2.4: Probability of collision resulting from varying the number of hashtables ($L = 1, 2, 10, 20$) for fixed hashcode length $K = 5$. The hash function family is $\mathcal{H}_{cosine}$. We can see that as the number of hashtables increase there is a higher probability of two data-points being close by colliding in at least one of the hashtable buckets, and a very low probability of more distant data-points colliding. This is expected as with more hashtables we are more likely to find a situation where none of the randomly generated hyperplanes separate the two data-points. This figure is adapted from a similar chart in Petrovic (2012).

The query time complexity is also dependent on L and K. For a single query data-point the retrieval cost can be characterised by $O(KDL) + O(1) + O(\frac{NDL}{2^K})$. This is made up of the *hashing time* (time spent generating the hashcodes) $O(KDL)$, the *lookup time* ($O(1)$ for a good hashtable implementation) and the *candidate test time* ($O(\frac{NDL}{2^K})$), which is the time taken to exhaustively compute the distance from the query to the colliding data-points and assuming a uniform distribution of data-points to buckets. As K is increased the hashing time will increase but the candidate test time will fall as the hash functions become more selective. Increasing the number of hashtables will increase both the hashing time and candidate test time with the benefit of increasing the probability that close-by data-points will collide in at least one bucket of a hashtable.

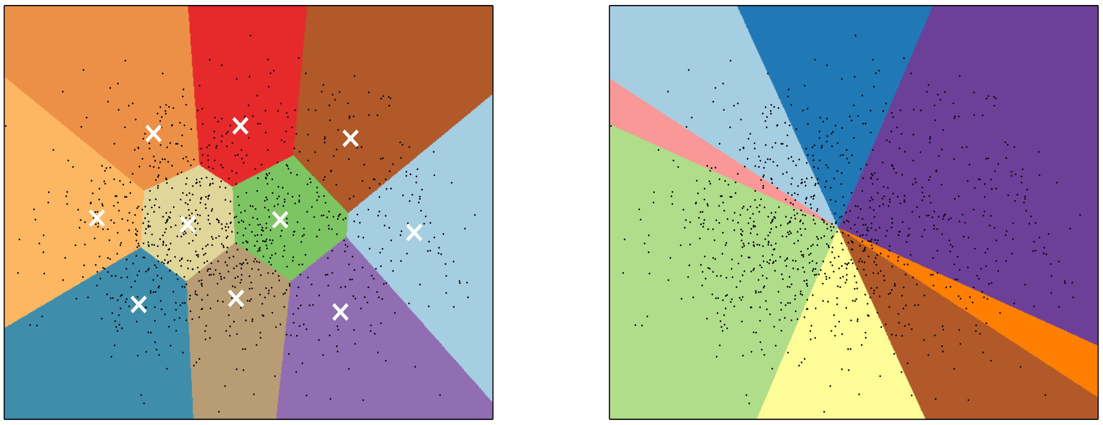
Equation 2.7 provides the foundation upon which the rest of this review and indeed thesis is formed. I propose novel formulations for defining the projection function

$\{p_k : \mathbb{R}^D \rightarrow \mathbb{R}\}$ and the quantisation function $\left\{q_k : \mathbb{R} \rightarrow \{0,1\}^B\right\}$ so that retrieval effectiveness is maximised, while also maintaining scalability of the algorithms to large datasets. More specifically I will firstly challenge the notion that a sign function is an optimal quantisation strategy for converting the real-valued projection in Equation 2.7 to binary (Section 2.5 and Chapters 4-5) and secondly, that randomly sampled hyperplanes produce optimal hashcodes (Section 2.6 and Chapter 6). On the latter point, it is well known that LSH hyperplanes tend to lack discriminative power with many hyperplanes (bits) and many hashtables being required for an adequate level of precision and recall (Wang et al. (2012)). This inefficiency is due to their *data-independent* nature where the hashing hyperplanes are generated without regard to the data distribution. This issue has recently stimulated research into hashing methods that learn hash functions adapted to the distribution of the data. I discuss state-of-the-art data-driven hash functions in Sections 2.5-2.6.

2.5 Quantisation for Nearest Neighbour Search

In this section I review previous related research in the field of binary quantisation for hashing-based ANN search. We saw in Section 2.4.1 that one of the two crucial steps in generating LSH-based binary hashcodes involves converting real-valued projections into binary bits. In this section I study in depth recently proposed algorithms for reducing the information loss resulting from the discretisation of real-numbers into binary, and specifically methods that attempt to do better than simply taking the sign function in Equation 2.7. I will attempt to put on a firm grounding exactly what I mean by better later in this section. Generally speaking, quantisation refers to the process of reducing the cardinality of a representation (such as numbers on the real-line) to a finite and discrete set of symbols (e.g. binary bits). Quantisation has been extensively studied particularly within the field of information theory (Gray and Neuhoff (2006)) and has also found wide engineering application given the impossibility of storing and manipulating numeric values to infinite precision. This review will necessarily only focus on quantisation methods that have been specifically used in hashing-based ANN search methods.

Two main categories of quantisation have been proposed for nearest neighbour search: *scalar* and *vector* quantisation, which are differentiated by whether the input and output of the quantisation is a scalar or a vector quantity. Scalar quantisation is frequently applied to quantise the real-values (projections) resulting from the dot prod-



(a) **Vector Quantisation** (b) **Scalar Quantisation**

Figure 2.5: In the context of nearest neighbour search two variations on quantisation are typically employed: vector quantisation (Figure (a)) partitions the feature space into Voronoi cells (Jegou et al. (2011)). Centroids are marked with a white cross while data-points are shown as black dots. The distance between query and database points is computed by determining the distance to their closest centroids. In contrast, scalar quantisation (Figure (b)) is frequently used to binarise a real-valued projection resulting from a dot product of a data-point with a hyperplane normal vector. The space is partitioned with multiple such hyperplanes and each usually contribute 1-bit to the final hashcode. The hashcode is effectively the index of the polytope-shaped region containing the associated data-point. The data-points appearing in this example are a 2D PCA projection of the CIFAR-10 image dataset.

uct of the feature vector of each data-point onto the normal vectors to a set of random hyperplanes partitioning the feature space (Figure 2.5). As we will discover in Section 2.5.1, each dot product yields a scalar value which is then subsequently quantised into binary $(0/1)$ by thresholding. In contrast, for vector quantisation, the feature representation of a data-point is associated with its nearest centroid, out of a set of centroids discovered by the k-means algorithm (Lloyd (1982)). In this way each input vector (data-point) is represented by a much smaller set of codebook vectors (centroids). K-means divides the space into Voronoi regions forming a more flexible data-driven partitioning. I illustrate the partitions formed by vector quantisation and a hyperplane based scalar quantisation in Figure 2.5. I will not cover vector quantisation any further in this thesis, but I will mention in passing that it has been found to be more effective for nearest neighbour search in Computer Vision tasks due to lower reconstruction error (Jegou et al. (2011)). The downside is the need to store a lookup table at test time to

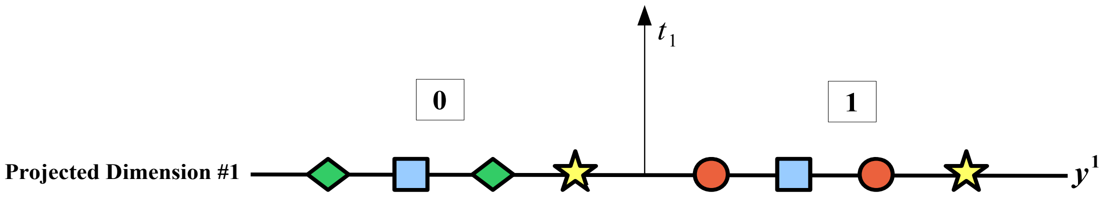


Figure 2.6: Single bit quantisation (SBQ) uses one threshold $t_1 \in \mathbb{R}$ to binarise a projected dimension: projected values (indicated by the coloured shapes) lower than the threshold (on the left) are assigned a '0' bit, while projected values greater than the threshold (on the right) are assigned a '1' bit.

read off inter-cluster Euclidean distances using the centroid indices[2]. This computation has been found to be 10-20 times slower than the Hamming distance computation between the binary hashcodes on standard datasets (He et al. (2013)). Scalar quantisation needs no such *decoding* step as the distances are computed directly from the hashcodes, an advantage that has proved beneficial in applications such as mobile product search (Feng (2012)). Only very recently have researchers attempted to combine the strengths of both approaches in a unified quantisation algorithm: the reader is encouraged to consult He et al. (2013) and references therein for more detail on interesting work in this direction.

In the context of hashing-based ANN search a scalar quantiser $\left\{q_k : \mathbb{R} \rightarrow \{0,1\}^B\right\}$ maps a real-valued projection $y_i \in \mathbb{R}$ to a single (Section 2.5.1) or multi-bit (Sections 2.5.3-2.5.4) binary codeword $\left\{\mathbf{c}_i = \{0,1\}^B | \mathbf{c}_i \in \mathcal{C}, i \in \{1,2,\ldots,T+1\}\right\}$ with T denoting the number of quantisation thresholds, B denoting the number of bits per projected dimension and $\mathcal{C}$ is the binary codebook. In this dissertation I follow Kong et al. (2012) by defining a *projected dimension* $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$ as the set of real-valued projections $\{y_i^k \in \mathbb{R}\}_{i=1}^{N_{trd}}$ of all data-points $[\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_{N_{trd}}]$ for a given hyperplane $\mathbf{h}_k$, where a projection $y_i^k \in \mathbb{R}$ is obtained by a dot product $y_i^k = \mathbf{w}_k^\intercal \mathbf{x}_i$. The quantisation function q_k binarises each projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$ independently by positioning one or more thresholds at selected points along the dimension. Projected values falling into a given thresholded region are assigned the codeword of that region. A simple illustration of this process is shown in Figure 2.6 where the projected dimension is shown as a line with a sampling of data-points (indicated by the coloured shapes) superimposed. In this toy example the quantiser uses a single threshold to partition the

[2]Another advantage of partitioning the space with hyperplanes is the exponential number (2^K) regions formed using just K-hyperplanes. Vector quantisation would require 2^K centroids for a similar partition granularity. Jegou et al. (2011) show how 2^K centroids can be learnt efficiently for large K using *product* quantisation.

Method	**Encoding**	**Optimisation**	**Thresholds (T)**	**Complexity**	**Section**
SBQ	0/1	Mean thresholding	1	$O(1)$	2.5.1
HQ	00/01/10/11	Spectral partitioning	1 and 2	$O(CN_{trd}^{+})$	2.5.2
DBQ	00/10/11	Squared error	2	$O(N_{trd} \log N_{trd})$	2.5.3
MHQ	NBC	1D K-means	3+	$O(2^B N_{trd})$	2.5.4

Table 2.1: Existing single (SBQ) and multi-threshold (HQ, DBQ, MHQ) quantisation schemes categorised along the three main dimensions of variability. NBC stands for Natural Binary Encoding and is explained in Section 2.5.4. C is the number of anchor points, N_{trd} is the number of training data-points, N_{trd}^{+} is the number of training data-points with positive projected value for the given projected dimension and B is the number of bits per projected dimension. Time complexity is for positioning thresholds along a single projected dimension.

projected dimension into two disjoint regions. The projections falling in the region below the threshold are given the codeword '0' while the projections falling in the region above the threshold are assigned the codeword '1'. The codebook for this example is $\{\mathbf{c}_i = \{0,1\} \,|\mathbf{c}_i \in \mathcal{C}, i \in \{1,2\}\}$. In this way, quantisation transforms projected values that live on the real-line into a discrete set of codewords from the specified codebook $\mathcal{C}$. More formally I denote as $\mathbf{t}_k = [t_1, t_2, \ldots, t_T]$ the set of threshold positions along a single projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$ where $t_i \in \mathbb{R}$ with $t_1 \leq t_2 \ldots \leq t_T$. The two extremities of a projected dimension are denoted as $t_0 = -\infty$ and $t_{T+1} = +\infty$. The thresholds $\{t_i \in \mathbb{R}\}_{i=1}^{T}$ partition a given projected dimension into T+1 disjoint regions $\mathbf{r}_i = \left\{y_j | t_{i-1} < y_j \leq t_i, y_j \in \mathbf{y}_k\right\}$ where $i \in \{1 \ldots T+1\}$. Most existing scalar quantisation schemes use $T = 2^B - 1$ thresholds for a budget of B bits per projected dimension. Each of the resulting $T+1$ thresholded regions $\left\{\mathbf{r}_i \subset \mathbf{y}^k\right\}_{i=1}^{T+1}$ are associated with a unique codeword $\mathbf{c}_i \in \mathcal{C}$.

The retrieval effectiveness resulting from quantisation is greatly affected by the selected codebook and the positioning of the quantisation thresholds (Moran et al. (2013a); Kong et al. (2012); Kong and Li (2012a)). The encoding scheme must ensure that the relative distances between the data-points in the input space are maintained in the resulting binary hashcodes. For example, if two data-points are distant in the original feature space then their assigned codewords should also be distant in Hamming space, and vice-versa for close data-points. Ideally the encoding scheme for the thresholded regions should impart a smooth, gradual increase in Hamming distance as the distance between the data-points in the original feature space increases. Correct positioning of the thresholds is also important as a threshold dividing an area dense in true nearest neighbours will result in related data-points falling into different regions and being assigned different codewords, increasing the Hamming distance of their resulting hashcodes. If the threshold positions and/or the encoding scheme are sub-optimal then the related data-points will be assigned hashcodes with large Hamming distance severely limiting the effectiveness of any hashing algorithm using that quantisation scheme. The state-of-the-art quantisation algorithms I review in this section all propose an encoding scheme and threshold optimisation algorithm that seek to faithfully preserve the relative distances between data-points in their corresponding binary hashcodes.

The previous paragraph hints at three key properties that can be used to distinguish and categorise existing methods of scalar quantisation[3]: the encoding scheme used to assign symbols to each thresholded region, the manner in which the threshold positions are determined, that is, whether a learning scheme is employed to optimise the positioning, and the number of thresholds allocated per dimension. In Table 2.1 I provide an overview of existing quantisation methods as categorised along these three dimensions of variability. In the following sections I describe these existing quantisation schemes in detail. Specifically, in Section 2.5.1 I introduce the traditional method of Single Bit Quantisation (SBQ) and in Sections 2.5.2-2.5.4 I describe the more recent multi-threshold quantisation schemes: Hierarchical Quantisation (HQ), Double Bit Quantisation (DBQ) (Section 2.5.3) and Manhattan Hashing Quantisation (MHQ) (Section 2.5.4).

2.5.1 Single Bit Quantisation (SBQ)

Single Bit Quantisation (SBQ) is the method of binarisation employed in the vast majority of existing hashing methods. A single threshold t_k partitions a projected dimension $\mathbf{y}^k$ into two regions, with a '0' bit assigned to projected values lower than the threshold and a '1' bit assigned to projected values equal to or greater than the threshold. More formally given a set of k hyperplane normal vectors $[\mathbf{w}_1 \ldots \mathbf{w}_K]$, the k^{th} hashcode bit for a data-point $\mathbf{x}_i$ is generated by SBQ as given in Equation 2.9.

[3] I will use the term *quantisation* to refer to scalar quantisation throughout the remainder of this thesis.

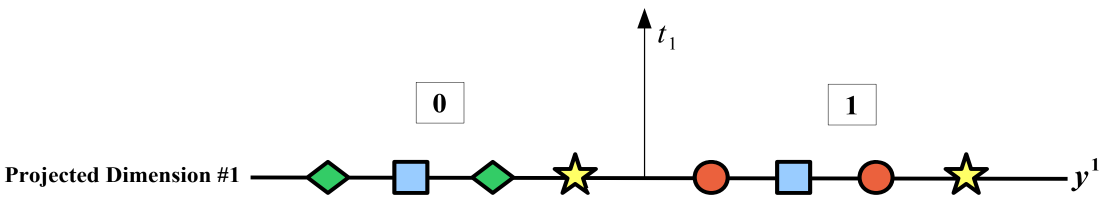


Figure 2.7: Single Bit Quantisation (SBQ) uses one threshold t_1 to binarise a projected dimension: projected values (indicated by the coloured shapes) lower than the threshold (on the left) are assigned a 0 bit, while projected values greater than the threshold (on the right) are assigned a 1 bit.

$$h_k(\mathbf{x}_i) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k)) \tag{2.9}$$

In this quantisation scheme each hyperplane contributes one bit towards the hashcode for a data-point. The data is typically *zero-centred* and the projected dimensions are thresholded at the mean ($t_k = \frac{1}{N_{trd}} \sum_{i=1}^{N_{trd}} \mathbf{w}_k^\intercal \mathbf{x}_i$). For zero centered data this equates to the threshold being placed directly at zero ($t_k = 0$). No learning mechanism is used to optimise the placement of the threshold in SBQ, although in some cases it might be placed at the median of the data distribution rather than at the mean. Given the absence of a threshold optimisation step SBQ is a computationally inexpensive operation requiring $O(1)$ time[4] for threshold learning and $O(1)$ time for encoding a novel query data-point. SBQ is further illustrated with a toy example in Figure 2.7.

The multi-threshold quantisation algorithms I describe in Section 2.5.2-2.5.4 all seek to overcome a fundamental limitation of SBQ which arises from the use of a single threshold for binarisation. In some cases SBQ may assign different bits to data-points that are located close together along a projected dimension, while data-points that are located much further apart can be assigned the same bits (Kong et al. (2012); Kong and Li (2012a); Moran et al. (2013a,b)). This is contrary to the fundamental objective of hashing in which close-by data-points should be assigned identical bits. Given this, it should be expected that this limitation of SBQ can lead to reduced retrieval effectiveness. I experimentally confirm that this is the case in Chapter 4. This problem with SBQ is easily illustrated by considering a hypothetical true nearest neighbour data-point pair in Figure 2.8. In this diagram the data-points a and b indicated by the yellow stars are very close to the threshold but lie on opposite sides. Even though both are close in the projected space they are assigned opposite bits, increasing the Ham-

[4]This increases to $O(N_{trd})$ time for threshold learning if the median is used as the threshold.

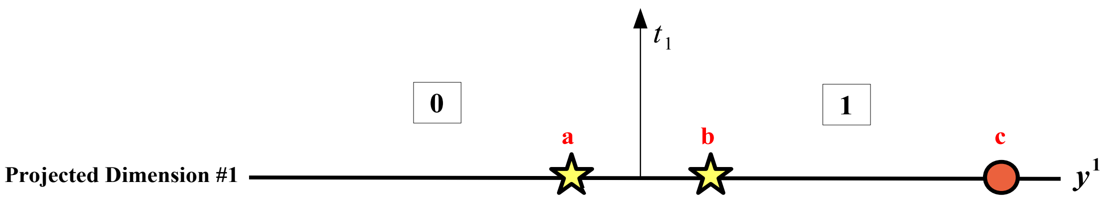
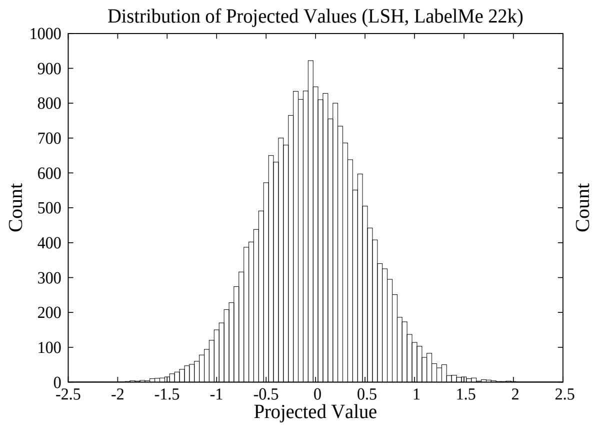


Figure 2.8: The problem with Single Bit Quantisation (SBQ): true nearest neighbours such points a,b are assigned different bits despite being close together along the projected dimension. Conversely points b,c located above the threshold, which are non-nearest neighbours, are assigned the same bit (1), even though they are more distantly spaced along the projected dimension.

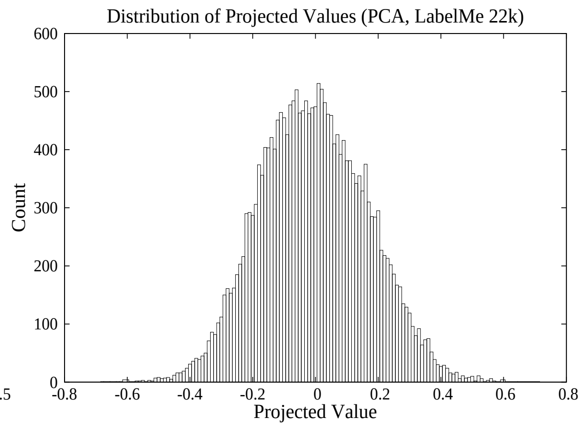
ming distance of their hashcodes. The hash function, by placing the projections of the pair a,b nearby along the projected dimension, has indicated that the corresponding data-points were close together (as deemed by our distance metric of interest) in the original feature space[5]. Despite this, SBQ assigns both opposite bits, effectively destroying the neighbourhood structure encoded in the projections. The opposite is true for the data-points b,c indicated by the yellow star and red circle located above the threshold. These non-nearest neighbours are far apart along the projected dimension, indicating that the hash function determined they were more distant in the original feature space. Nevertheless, SBQ has assigned the same bit to both data-points b,c ensuring their resulting hashcodes are closer together in terms of Hamming distance.

Unfortunately, this issue is likely to surface often in practice given that vanilla SBQ places a threshold directly at zero and the highest point density along a projected dimension also usually happens to be in the region around zero. This pattern is true for many projection functions commonly employed in practice (Figure 2.9). Partitioning a projected dimension into multiple regions, and assigning each region a multi-bit encoding is an effective means of overcoming this issue with SBQ. The *modus-operandi* of all multi-threshold quantisation schemes, including my novel contributions presented in Chapter 4 and Chapter 5, is maximal preservation of the neighbourhood structure encoded in the projected dimension through a multi-bit codebook and a threshold optimisation algorithm. I will now discuss one of the first proposed multi-threshold quantisation schemes in Section 2.5.2.

[5]We are of course relying here on the hash function placing data-points that are close-by in the original feature space close by along the projected dimension. Randomised LSH projection functions guarantee this in expectation while other projection functions seek to explicitly learn the hyperplanes so that related data-points are encouraged to have similar projections. I cover the latter data-dependent methods in Section 2.6.



(a) **Locality Sensitive Hashing (LSH)**



(b) **Principal Component Analysis (PCA)**

Figure 2.9: Distribution of projected values for two randomly chosen LSH (Figure (a)) and PCA (Figure (b)) projected dimensions on the LabelMe 22k image dataset (Torralba et al. (2008)). The images in this dataset are encoded as GIST features (Oliva and Torralba (2001)). The region of highest projected value density is typically around zero, as is clearly the case for these two dimensions. The Double Bit Quantisation algorithm (DBQ) Kong and Li (2012a) explicitly avoids placing a threshold close to zero as, given the high density of points in that region, this is likely to separate many true nearest neighbours. DBQ is described in Section 2.5.3.

2.5.2 Hierarchical Quantisation (HQ)

Liu et al. (2011) were the first to introduce the concept of multi-threshold quantisation for hashing in which a single hyperplane contributes multiple bits to the hashcode. Their quantisation algorithm, dubbed Hierarchical Quantisation (HQ), was introduced as a means of quantising projections resulting from their Anchor Graph Hashing (AGH) model. It uses only $\lfloor K/2 \rfloor$ of the available hyperplanes, assigning two bits per hyperplane. I describe the projection learning component of AGH in detail in Section 2.6.3.4, while in this section I focus solely on the quantisation algorithm assuming that we have already obtained the desired projected dimensions $\{\mathbf{y}^k \in \mathbb{R}^{N_{trd}}\}_{k=1}^{K}$. HQ consists of two steps performed in a sequence: in the first step traditional SBQ is applied (Section 2.5.1), thresholding a given projected dimension $\mathbf{y}^k$ at zero ($t_1 = 0$). Step one produces the first bit of the double-bit encoding for a hyperplane. In the second step the projected dimension is quantised again, this time using two new thresholds (t_2, t_3) that further partition the two regions formed by SBQ at the first step.

The two thresholds (t_2, t_3) are jointly optimised so as to minimise the number of related data-points falling on opposite sides of the dividing threshold resulting from applying SBQ in the first step. Both quantisation steps are illustrated in Figure 2.10. Liu et al. (2011) formulate the threshold optimisation as a graph partitioning problem on the graph Laplacian $\mathbf{L} = \mathbf{I} - \hat{\mathbf{S}}$ of the low-rank *approximate* adjacency matrix $\hat{\mathbf{S}}$[6]. The neighbourhood structure is encoded by $\hat{\mathbf{S}}$, where $\hat{S}_{ij} > 0$ indicates that i and j are neighbours, and $\hat{S}_{ij} = 0$ indicates they are not. $\hat{\mathbf{S}}$ is approximate in the sense that it is not constructed by computing the N_{trd}^2 distances between the N_{trd} data-points, but rather is constructed from an anchor graph $\mathbf{Z}$, a sparse matrix that holds the similarities between the N_{trd} data-points and a small set of C anchor points where $C \ll N_{trd}$ (Equation 2.10):

$$Z_{ij} = \begin{cases} \dfrac{exp(-d^2(\mathbf{x}_j, \mathbf{c}_i)/\gamma)}{\sum\limits_{i' \in \langle j \rangle} exp(-d^2(\mathbf{x}_j, \mathbf{c}_{i'}))/\gamma)} & \text{if } i \in \langle j \rangle \\ 0 & otherwise \end{cases} \tag{2.10}$$

where γ is the kernel bandwidth, $\{d(.,.) : \mathbb{R}^D \times \mathbb{R}^D \to [0,1]\}$ is any distance function of interest such as the L_2-norm and $\langle j \rangle \in \{1 \ldots R\}$ are the indices of the $R \ll C$ nearest anchors to $\mathbf{x}_j$ under distance metric $d(.,.)$. The C anchor points $\{\mathbf{c}_i \in \mathbb{R}^D\}_{i=1}^C$ can be found by running the k-means algorithm over data-points in a training dataset. Using the anchor graph, the adjacency matrix $\hat{\mathbf{S}}$ can be computed as $\hat{\mathbf{S}} = \mathbf{Z}\Lambda^{-1}\mathbf{Z}^\intercal$ where $\Lambda_{kk} = \sum_{i=1}^{N_{trd}} Z_{ik}$. This latter expression approximates the affinity between data-points $\mathbf{x}_i, \mathbf{x}_j$ as the inner product between their individual affinities to the C centroids. Compared to the full adjacency matrix, $\hat{\mathbf{S}}$ is sparse and low-rank which brings computational advantages when extracting the required graph Laplacian eigenvectors (Section 2.6.3.4). Liu et al. (2011) construct an eigenvalue problem involving $\mathbf{Z}$ to solve for the K graph Laplacian eigenvectors $\mathbf{y}^k$ of the approximate adjacency matrix $\hat{\mathbf{S}}$. In the context of AGH these eigenvectors are the projected dimensions that are thresholded to yield the hashcodes of the training data-points (Equation 2.11).

$$h_k(\mathbf{x}_i) = \begin{cases} \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i - t_2)), & \text{if} \quad \mathbf{w}_k^\intercal \mathbf{x}_i \geq 0 \\ \frac{1}{2}(1 + sgn(-\mathbf{w}_k^\intercal \mathbf{x}_i + t_3)), & \text{if} \quad \mathbf{w}_k^\intercal \mathbf{x}_i < 0 \end{cases} \tag{2.11}$$

Binarising a graph Laplacian eigenvector has the effect of partitioning the graph

[6]The rank of a matrix is the number of linearly independent rows or columns.

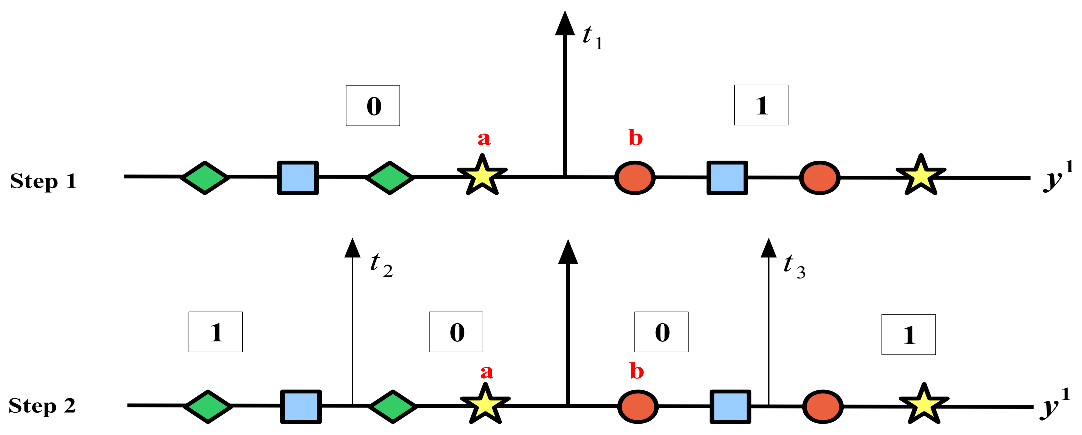


Figure 2.10: Illustration of the Hierarchical Quantisation (HQ) algorithm of Liu et al. (2011). I use the data-points a (yellow star) and b (red circle) as examples. The quantisation proceeds in two steps: in the first step SBQ thresholds the projected dimension into two regions generating the first bit of the two bit encoding for the data-points. For data-point a this is '0' and for data-point b this is '1'. In the second step the regions formed by SBQ are further partitioned with t_2, t_3 using a dynamic threshold optimisation algorithm. Data-point a is assigned '0' again and b is now assigned '0', yielding hashcodes '00' and '10', respectively. Nearby data-points falling on opposite sides of the SBQ threshold in Step 1 are therefore more likely to have the same bit assigned in Step 2, which is the case for our two example data-points. This is the central tenet of the HQ algorithm.

encoded by $\hat{\mathbf{S}}$ into two groups, with each of the K eigenvectors forming a different bi-partitioning of the graph (Shi and Malik (2000)). In the context of hashing-based approximate nearest neighbour search, the hope is that many of these graph cuts will result in true nearest neighbours being within the same partition. The eigenvectors with the highest eigenvalues are generally unreliable and do not produce an effective partitioning of the graph (Shi and Malik (2000)). This observation motivates the creation of the two-step quantisation algorithm of Liu et al. (2011) in which the lowest eigenvectors are responsible for generating most of the hashcode bits. If we denote $\mathbf{y}^{k+}$ as the positive projected values of projected dimension $\{\mathbf{y}^{k+} \in \mathbb{R}^{N_{trd}} | y_i^k \geq t_1\}$, and $\{\mathbf{y}^{k-} \in \mathbb{R}^{N_{trd}} | y_i^k < t_1\}$ the negative projected values, the objective of the second level threshold optimisation is to minimise Equation 2.12

$$\begin{aligned} &\operatorname{argmin}_{t_2,t_3} \quad \mathbf{f}^\mathsf{T}\mathbf{L}\mathbf{f} \\ &\quad \text{where } \mathbf{f} = \begin{bmatrix} \mathbf{y}^+ - t_2\mathbf{1}_1 \\ -\mathbf{y}^- + t_3\mathbf{1}_2 \end{bmatrix} \\ &\text{subject to } \mathbf{1}^\mathsf{T}\mathbf{f} = 0 \end{aligned} \tag{2.12}$$

Intuitively the objective function is attempting to position thresholds t_2, t_3 so that two conditions are met. Firstly, connected nodes in $\hat{\mathbf{S}}$, that is true nearest neighbours, stay as close as possible along the projected dimension (as $\mathbf{f}^\mathsf{T}\mathbf{L}\mathbf{f} = \sum_{ij} \hat{S}_{ij}(f_i - f_j)^2$), and secondly, there is an equal number of opposing bits ('0' and '1's) when the projected dimension is binarised with thresholds t_2, t_3. This latter balance constraint ($\mathbf{1}^\mathsf{T}\mathbf{f} = 0$) has previously been shown to maximise the information captured by the bits (Weiss et al. (2008)). Liu et al. (2011) demonstrate that by setting to zero the derivatives of Equation 2.12, the two thresholds t_2, t_3 minimising Equation 2.12 can be computed in closed form.

While Liu et al. (2011) find their multi-threshold quantisation algorithm more effective than SBQ it suffers from lack of generality to other projection functions, being entirely tied to the quantisation of graph Laplacian eigenvectors. The computational complexity of solving for t_2, t_3 is approximately $O(CN^+_{trd})$ (Liu et al. (2011)), where N^+_{trd} denotes the number of positive projected values constituting $\mathbf{y}^{k+}$. Given the learnt thresholds the time taken to generate a bit for a novel query point is $O(1)$. HQ effectively front loads the available bit budget onto the lowest graph Laplacian eigenvectors. Liu et al. (2011) demonstrate that only using half the number of eigenvectors and assigning each with two bits yields higher retrieval effectiveness than using all available eigenvectors and assigning each a single bit. Typically, the intrinsic dimension of many datasets of interest is low and so the lower graph Laplacian eigenvectors (those with the smallest eigenvalues) are likely to capture most of the neighbourhood structure, with the higher eigenvectors being more informative of the input space. This insight is the seed that sparked the recent interest in multi-threshold quantisation algorithms for ANN search and is the inspiration behind our novel contributions in Chapter 5. I will examine subsequent research contributions in this area in chronological order continuing next to the Double Bit Quantisation (DBQ) algorithm of Kong and Li (2012a).

2.5.3 Double Bit Quantisation (DBQ)

Double-Bit Quantisation (DBQ) (Kong and Li (2012a)) allocates two thresholds per projected dimension and assigns two bits to the three resulting thresholded regions. Unlike HQ (Section 2.5.2) it has the useful advantage of not being tied to any specific projection function. For a K-bit hashcode DBQ therefore only uses $\lfloor K/2 \rfloor$ of the number of hyperplanes as SBQ. DBQ uses the binary encoding scheme illustrated in Figure 2.11 which ensures that any two adjacent regions only differ by unit Hamming distance. This property is crucial if the relative distances between the data-points are to be maintained in the underlying binary encoding, a key requirement for maximising retrieval effectiveness. DBQ also proposes a novel adaptive thresholding algorithm for finding an optimal setting of the quantisation thresholds t_1, t_2. Given a particular instantiation of the quantisation thresholds t_1, t_2, three sets $\mathbf{r}_1$, $\mathbf{r}_2$, $\mathbf{r}_3$ are defined each containing the projected values falling within the corresponding region, that is: $\mathbf{r}_1 = \{y_i | y_i \leq t_1, y_i \in \mathbf{y}_k\}$, $\mathbf{r}_2 = \{y_i | t_1 < y_i \leq t_2, y_i \in \mathbf{y}_k\}$, $\mathbf{r}_3 = \{y_i | t_2 < y_i, y_i \in \mathbf{y}_k\}$. The objective function $\mathcal{J}_{dbq}$ of DBQ is to minimise the sum of squared Euclidean distances of the projected values falling within the three thresholded regions (Equation 2.13)

$$\mathcal{J}_{dbq}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) = \sum_{y_i \in \mathbf{r}_1} (y_i - \mu_1)^2 + \sum_{y_j \in \mathbf{r}_2} (y_j - \mu_2)^2 + \sum_{y_l \in \mathbf{r}_3} (y_l - \mu_3)^2 \tag{2.13}$$

where μ_i denotes the mean of the projected values in region $\mathbf{r}_i$. As the area of highest point density along a projected dimension is in the region of zero (Figure 2.9), DBQ avoids placing a threshold at zero by setting $\mu_2 = 0$ enforcing the property that $t_1 < 0$ and $t_2 > 0$. Given this $\mathcal{J}_{dbq}$ can then be simplified as in Equation 2.16 (Kong and Li (2012a))

$$\begin{aligned}
\mathcal{J}_{dbq}(\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) &= \sum_{y_i \in \mathbf{y}_k} y_i^2 - 2 \sum_{y_i \in \mathbf{r}_1} y_i \mu_1 + \sum_{y_i \in \mathbf{r}_1} \mu_1^2 - 2 \sum_{y_l \in \mathbf{r}_3} y_l \mu_3 + \sum_{y_l \in \mathbf{r}_3} \mu_3^2, && (2.14) \\
&= \sum_{y_i \in \mathbf{y}_k} y_i^2 - |\mathbf{r}_1| \mu_1^2 - |\mathbf{r}_3| \mu_3^2, && (2.15) \\
&= \sum_{y_i \in \mathbf{y}_k} y_i^2 - \frac{(\sum_{y_i \in \mathbf{r}_1} y_i)^2}{|\mathbf{r}_1|} - \frac{(\sum_{y_i \in \mathbf{r}_3} y_i)^2}{|\mathbf{r}_3|} && (2.16)
\end{aligned}$$

where $|\mathbf{r}_i|$ is the number of projected values (data-points) in region $\mathbf{r}_i$. Given that $\sum_{y_i \in \mathbf{y}_k} y_i^2$ is a constant minimising $\mathcal{J}_{dbq}$ is equivalent to maximising $\mathcal{J}_{dbq}^{'}$ (Equation 2.17).

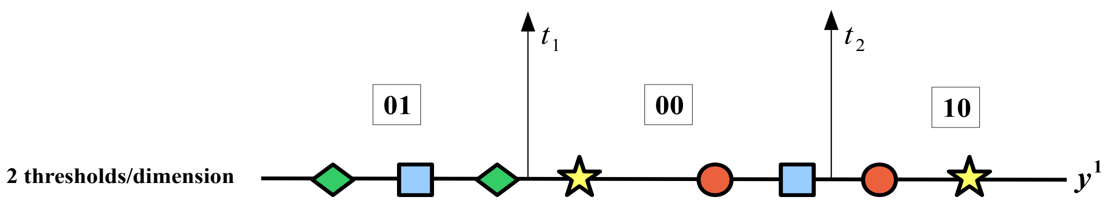


Figure 2.11: Double Bit Quantisation allocates two thresholds t_1, t_2 to binarise a projected dimension. The resulting three thresholded regions are assigned the two-bit encoding shown. This encoding ensures that adjacent regions are only separated by a Hamming distance of 1.

$$\mathcal{J}^{'}_{dbq}(\mathbf{r}_1, \mathbf{r}_3) = \frac{(\sum_{y_i \in \mathbf{r}_1} y_i)^2}{|\mathbf{r}_1|} + \frac{(\sum_{y_j \in \mathbf{r}_3} y_j)^2}{|\mathbf{r}_3|} \tag{2.17}$$

The objective function $\mathcal{J}^{'}_{dbq}$ is maximised by the adaptive thresholding strategy presented in Algorithm 3. Algorithm 3 initialises the thresholds t_1, t_2 to values close to zero, and then gradually moves both thresholds apart: t_1 is moved towards negative infinity ($-\infty$) while t_2 is moved towards positive infinity ($+\infty$). The objective function $\mathcal{J}^{'}_{dbq}$ is evaluated at every projected value along the projected dimension. In tandem to this the algorithm attempts to maintain the invariant that the mean of region $\mathbf{r}_2$ is zero i.e. $\mu_2 = 0$ (Line 9), which ensures that neither threshold partitions the densest region of the projected dimension located around zero. The thresholds are moved while maintaining this property by gradually shifting data-points from regions $\mathbf{r}_1$ and $\mathbf{r}_3$ into $\mathbf{r}_2$ (Lines 8-13): if the sum of projected values in $\mathbf{r}_2$ is below zero then a positive projected value from $\mathbf{r}_3$ is moved into $\mathbf{r}_1$ to increase the sum towards zero, and vice-versa. The objective function $\mathcal{J}^{'}_{dbq}$ is then computed (Line 15) on the new regions $\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$. If there is an increase in $\mathcal{J}^{'}_{dbq}$ the thresholds t_1, t_2 are updated to be the largest projected values in $\mathbf{r}_1$ and $\mathbf{r}_2$, which now define the new boundaries between the regions. The algorithm terminates when all data-points have been moved into region $\mathbf{r}_2$. Note that as all data-points along the projected dimension are exhaustively examined DBQ guarantees to find t_1, t_2 that lead to the global maximum of $\mathcal{J}^{'}_{dbq}$.

The implicit assumption made by DBQ is that the hash functions will minimise the squared Euclidean distance between true nearest neighbours in the low-dimensional projected space, which equates to the projected values of any two true nearest neighbours having low squared Euclidean distance along a given projected dimension. If this assumption is correct then a clustering of the one-dimensional projected dimension based on a squared error criterion will result in more true nearest neighbours being

Algorithm 3: Double Bit Quantisation (Kong and Li (2012a))

```
Input: Projected values $\mathbf{y}^k \in \mathbb{R}^N$ resulting from projection onto normal vector
       $\mathbf{w}_k \in \mathbb{R}^D$ of hyperplane $\mathbf{h}_k \in \mathbb{R}^D$
Output: Optimised thresholds $t_1 \in \mathbb{R}$, $t_2 \in \mathbb{R}$
1  $\mathbf{r}_1 \leftarrow \{y_i | y_i \leq 0, y_i \in \mathbf{y}^k\}$
2  $\mathbf{r}_2 \leftarrow \emptyset$
3  $\mathbf{r}_3 \leftarrow \{y_i | y_i > 0, y_i \in \mathbf{y}^k\}$
4  Sort (ascending) projected-values in $\mathbf{r}_1$
5  Sort (ascending) projected-values in $\mathbf{r}_3$
6  $J_{max} \leftarrow 0$
7  $i \leftarrow 1$
8  while $\mathbf{r}_1 \neq \emptyset$ or $\mathbf{r}_3 \neq \emptyset$ do
9      if $(sum(\mathbf{r}_2) \leq 0)$ then
10         $\mathbf{r}_2 \leftarrow \mathbf{r}_2 \cup min(\mathbf{r}_3)$          // Remove minimum value in $\mathbf{r}_3$
11     else
12         $\mathbf{r}_2 \leftarrow \mathbf{r}_2 \cup max(\mathbf{r}_1)$          // Remove maximum value in $\mathbf{r}_1$
13     end
14     $i \leftarrow i + 1$
15     $J \leftarrow \mathcal{J}'_{dbq}(\mathbf{r}_1, \mathbf{r}_3)$                    // Equation 2.17
16     if $(J > J_{max})$ then
17         $t_1 \leftarrow max(\mathbf{r}_1)$
18         $t_2 \leftarrow max(\mathbf{r}_2)$
19         $J_{max} \leftarrow J$
20     end
21 end
22 return $t_1, t_2$
```

assigned similar hashcodes (given that they will end up in the same thresholded region) versus an entirely random threshold setting. While Kong and Li (2012a) demonstrate that this is a reasonable assumption in practice, I will show in Chapter 4 that it is far from optimal and significantly improved retrieval effectiveness can be attained with a semi-supervised objective that does not entirely rely on the quality of the hash function that produces the projections. The threshold learning time complexity of DBQ is $O(N_{trd} \log N_{trd})$ which arises from the pre-processing step that sorts the projected values in regions $\mathbf{r}_1$ and $\mathbf{r}_3$ (Lines 4 and 5). DBQ has $O(1)$ time complexity when using the learnt thresholds to generate a bit for a novel query data-point.

2.5.4 Manhattan Hashing Quantisation (MHQ)

A primary disadvantage of both HQ and DBQ are their arbitrary restriction to two bits per projected dimension. Kong et al. (2012) explored the effect of introducing more bits per projected dimension in their Manhattan Hashing Quantisation (MHQ) model, which until the multi-threshold quantisation algorithms I present in Chapters 4-5, constituted the state-of-the-art in the field. MHQ permits an arbitrary allocation of bits, where for B bits per projected dimension $2^B - 1$ thresholds are used to partition the dimension into disjoint regions. To generate a hashcode of length K MHQ uses $\lfloor K/B \rfloor$ hyperplanes. In a similar manner to DBQ, MHQ introduces a new encoding scheme and threshold optimisation algorithm, both designed to increase the preservation of the relative distance between the data points in the resulting hashcodes. I describe both contributions in this section.

The binary encoding scheme advocated by MHQ is illustrated in Figure 2.12. Each region is encoded using natural binary encoding (NBC). The NBC codebook for each region is simply obtained by proceeding from left-to-right along the projected dimension starting with the region closest to $t_0 = -\infty$ and converting the integer index starting at zero (and incremented by one for each region) to its corresponding NBC. For example, in Figure 2.12 to obtain the NBC for the third region from the left for the bottom most projected dimension we convert the integer '2' to '010'. Under the constraint of Hamming distance it is clear that the NBC encoding scheme *does not* preserve the relative distance between the data-points. This is easily seen if we examine the encoding for the eight regions induced by setting seven thresholds along a projected dimension (Figure 2.12). The encoding for the fourth region from the left is 011, while the encoding for the adjacent region to the right is 100. The Hamming distance between these two regions is 3 despite both being adjacent, while the Hamming distance between region eight (111) - which is much further along the projected dimension - is only one. In effect this means that any data-points which are projected far apart along the projected dimension - and which are presumably far apart in the original feature space - will be much closer together in the Hamming space, than data-points that were projected close by along the projected dimension. Hashcodes generated with this encoding and compared using Hamming distance will yield poor quality hashcodes and low retrieval ef-

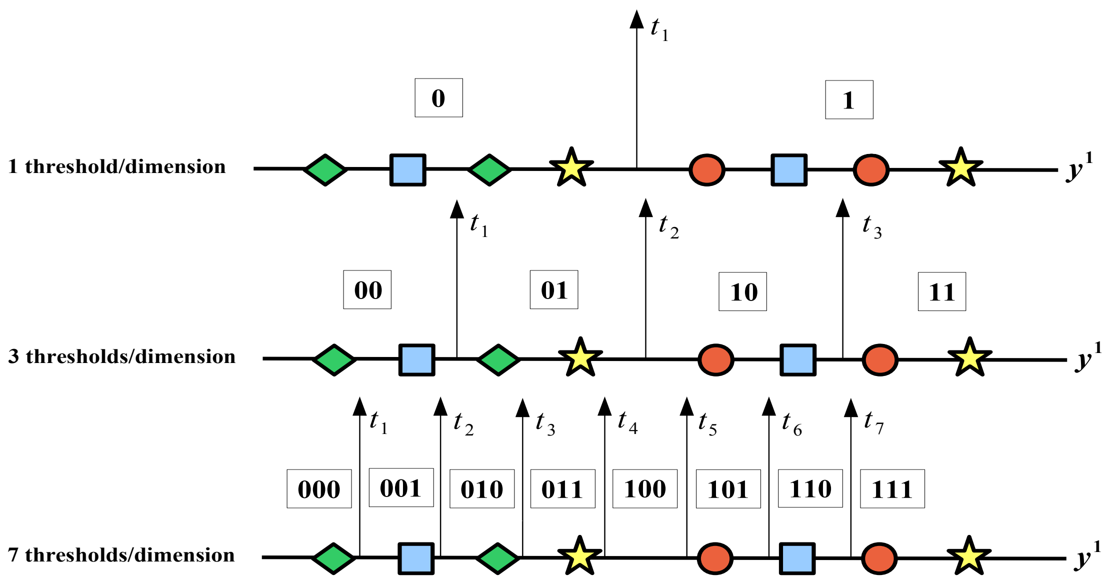


Figure 2.12: Manhattan Hashing Quantisation (MHQ) assigns $T = 2^B - 1$ thresholds per dimension, where B is the number of bits allocated per dimension. The encoding scheme for the thresholded regions is natural binary code (NBC). Each region from the left to the right is assigned an integer starting at 0 (on the left) and ending at $2^B - 1$ for the far right region. This integer index is converted to its equivalent NBC giving the codeword for that region.

fectiveness. To mitigate this issue Kong et al. (2012) propose taking the *Manhattan distance* between the integer index corresponding to a given NBC codeword, rather than the Hamming distance between the corresponding NBC codewords[7]. To illustrate how this method works I will consider the example given by Kong et al. (2012). Imagine we have generated the hashcode 000100 for data-point 1 and the hashcode 110000 for data-point 2. If $B = 2$, the Manhattan distance $\left\{d_{MHQ}(.,.) : \{0,1\}^K \times \{0,1\}^K \rightarrow \mathbb{Z}_+\right\}$ between the codewords is computed as in Equation 2.18

$$\begin{aligned} d_{MHQ}(000100, 110000) &= d_M(00, 11) + d_M(01, 00) + d_M(00, 00) \\ &= 3 + 1 + 0 \\ &= 4 \end{aligned} \tag{2.18}$$

[7]I note in passing that *binary reflected Gray coding* would *not* be suitable as a binary codebook for nearest neighbour search. Gray coding has the special property that adjacent codewords differ by unit Hamming distance which has proved beneficial for enabling error correction in digital communication over analog channels (Gray (1953)). Gray coding is unsuitable for nearest neighbour search, however, due to the fact that codewords for data-points located much further apart can also have unit Hamming distance therefore breaking the neighbourhood structure.

If the number of bits per dimension $B = 3$ then the computation proceeds as in Equation 2.19.

$$\begin{aligned} d_{MHQ}(000100, 110000) &= d_M(000, 110) + d_M(100, 000) \\ &= 6 + 4 \\ &= 10 \end{aligned} \tag{2.19}$$

Computing the Manhattan distance between the integer indices of each region leads to remarkable increases in retrieval effectiveness as demonstrated in Kong et al. (2012). This is primarily due to the perfect preservation of the relative distance between the data-points: the codeword for each adjacent thresholded region is a unit Manhattan distance apart and there is a smooth increase in the Manhattan distance between any two regions the further apart they are along the projected dimension. Furthermore, this encoding scheme generalises easily to any desired number of thresholds because we are simply taking the integer index of each region. The obvious downside to computing the Manhattan distance between the integer indices of the thresholded regions is the slower distance computation versus computing the Hamming distance. On most modern processors the Hamming distance can be efficiently computed using a bitwise `XOR` between the hashcodes followed by a native `POPCOUNT` instruction which counts the number of bits set to one. It is not clear in Kong et al. (2012) whether or not the Manhattan distance will become a bottleneck on large datasets of millions of data points and dimensions. Some authors have recently offered evidence that this may indeed be the case (Wang et al. (2015)) by showing that the Manhattan distance requires substantially more atomic operations on the CPU than the Hamming distance. In Chapter 4, I mitigate this concern by introducing a more general quantisation model that is effective with a binary encoding scheme under the Hamming distance metric in addition to the more recently proposed MHQ NBC encoding scheme coupled with the Manhattan distance metric.

The MHQ threshold optimisation algorithm is straightforward: k-means (Lloyd (1982)) with 2^B centroids $\{c_i \in \mathbb{R}\}_{i=1}^{2^B}$ is used to cluster the projected dimension. The corresponding $2^B - 1$ thresholds $\{t_i \in \mathbb{R}\}_{i=1}^{2^B-1}$ are computed from the centroids by taking the midpoint between adjacent centroids (Equation 2.20).

$$t_i = \frac{(c_i + c_{i+1})}{2} \tag{2.20}$$

MHQ requires $O(2^B N_{trd})$ time for threshold learning along a single projected dimension and $O(1)$ time to generate a bit for a novel query point with the learnt thresholds.

2.5.5 A Link to the Discretisation of Continuous Attributes

It is interesting to consider briefly how this research area relates to the well-studied area of *discretisation of continuous attributes* in the field of machine learning (Dougherty et al. (1995), Garcia et al. (2013)). Several well-known machine learning models such as Naïve Bayes (Bishop (2006); Yang and Webb (2009)) often have continuous attributes transformed into nominal attributes by discretisation prior to learning. The mechanism by which this continuous to discrete transformation is performed shares many similarities to the quantisation process in the field of multi-threshold quantisation for hashing. More specifically the attributes (dimensions) are partitioned with a set of cut-points (thresholds) forming a non-overlapping division of the continuous domain. In a similar manner to the quantisation algorithms I discussed in this section the real-valued numbers within each thresholded region are assigned the corresponding discrete symbol representing that region. These discrete symbols must be binary for the quantisation algorithms studied in this section but need not be for the discretisation algorithms found in machine learning. The discretisation literature is broad and varied and proposes a wealth of algorithms for learning the cut-points, ranging from unsupervised (simply place the cut-points at equal intervals) through to supervised (Fayyad and Irani (1993)) and multivariate (each attribute is discretised jointly) (Mehta et al. (2005), Kerber (1992)). Given the maturity of the discretisation research field I believe that there is significant potential for these already established ideas to inform the design of future scalar quantisation algorithms for hashing.

2.5.6 A Brief Summary

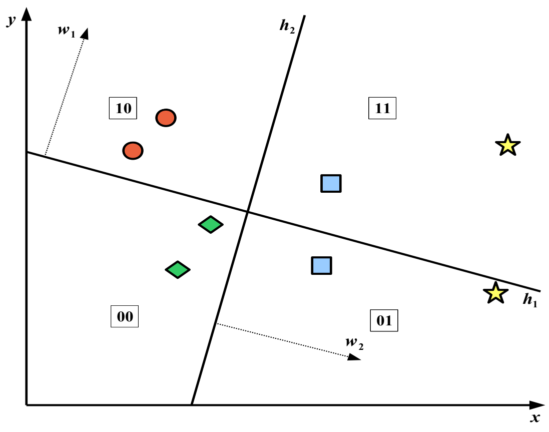
In this section I introduced four recently proposed algorithms for scalar quantisation in the context of hashing-based ANN search. Each method takes a series of real-valued projections and outputs binary bits which are concatenated to form the hashcodes for the data-points. Each algorithm is similar in the sense that one or more thresholds are used to perform the binarisation: if a value is above or below a threshold it is assigned a codeword (single bit or multiple bits) of the associated region so formed. Single Bit Quantisation (SBQ) is the standard method of quantisation used by most previous hashing models (Section 2.5.1). SBQ positions a single threshold, typically at zero along a projected dimension. SBQ has the advantages of being simple and computationally efficient, but as I argued, it can lead to high quantisation errors (related data-points being assigned different bits). The multi-threshold quantisation algorithms, Hierarchical Quantisation (HQ) (Section 2.5.2), Manhattan Hashing Quantisation (MHQ) (Section 2.5.4) and Double Bit Quantisation (DBQ) (Section 2.5.3) all seek to address this issue with SBQ using novel encoding schemes and threshold optimisation algorithms. The manner in which the thresholds are optimised varied widely with each algorithm: HQ relies on a spectral graph partitioning objective, while MHQ and DBQ optimise objections related to squared error and variance minimisation. The encoding schemes also differ significantly between the three multi-threshold algorithms, but all are designed so that the relative distance between the data-points is maximally preserved in the resulting hashcodes. I now turn our attention to a family of methods in Section 2.6 that are able to generate the projections that we have just quantised.

2.6 Projection for Nearest Neighbour Search

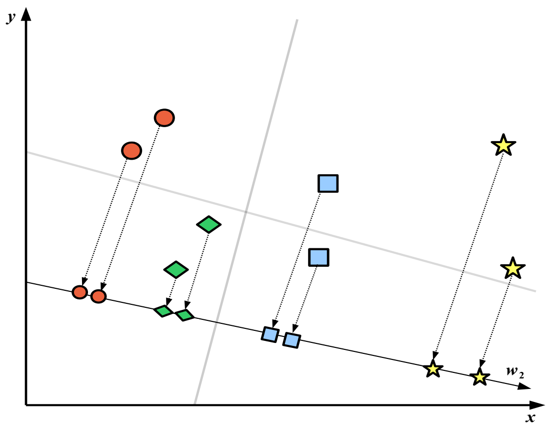
In Section 2.4.1, I identified two main steps - projection and quantisation - that are used to generate similarity preserving hashcodes in the context of Locality Sensitive Hashing (LSH). I discussed how both steps taken together and performed in a sequence effectively check which sides of a set of hyperplanes a data-point falls, appending a ‘1’ to the hashcode if a point falls on one side of a given hyperplane and a ‘0’ otherwise. In Section 2.5 I reviewed prior art that focused solely on improving the quantisation step. The quantisation algorithms I examined attempt to better preserve the neighbourhood structure between the data-points during binarisation, improving upon simply taking the sign of the projections in Equation 2.21

$$h_k(\mathbf{x}_i) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k)) \tag{2.21}$$

where $\mathbf{w}_k \in \mathbb{R}^D$ is the hyperplane normal vector and $t_k \in \mathbb{R}$ is the quantisation threshold. Equation 2.21 is the popular linear hash function adopted in most hashing research. I discussed in Section 2.5 that, apart from Anchor Graph Hashing (Liu et al. (2011)), most quantisation models operate independently of the projection stage and assume that the projections to be binarised have already been generated by an existing projection method. In this section I review the equally important step of *projection* and focus



(a) **Feature space partitioning**



(b) **Projection onto normal vector $\mathbf{w}_2$**

Figure 2.13: Illustration of the projection operation. Methods for projection partition the feature space with a set of hyperplanes. In Figure (a) I show a partitioning of a two dimensional feature space induced by two hyperplanes $\mathbf{h}_1$ and $\mathbf{h}_2$. To determine the bucket index or hashcode of a data-point it is necessary to project the data-points onto the normal vectors ($\mathbf{w}_1$, $\mathbf{w}_2$) followed by binary quantisation. In Figure (b) I show geometrically the result of projection onto normal vector $\mathbf{w}_2$. The resulting projections form projected dimension $\mathbf{y}^2 \in \mathbb{R}^{N_{trd}}$. Section 2.6 examines existing work that seek to position the hyperplanes so that many true nearest neighbours end up close to each other along the resulting projected dimensions.

on algorithms that seek to generate the projections in a way that preserves the relative distances between the data-points along the resulting projected dimensions. In the case of the linear hash function this is equivalent to positioning a set of K hyperplanes throughout the input feature space in such a way that similar data-points are likely to fall within the same polytope-shaped region. These regions constitute the hashtable buckets for indexing and retrieval. To generate a projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$ from a hyperplane $\mathbf{h}_k \in \mathbb{R}^D$ the data-points $\{\mathbf{x}_i \in \mathbb{R}^D\}_{i=1}^{N_{trd}}$ are projected onto the normal vector $\mathbf{w}_k \in \mathbb{R}^D$ using a dot product operation $\mathbf{w}_k^\intercal \mathbf{x}_i$. In Figure 2.13, I show geometrically the effect of the dot product and how a projected dimension is formed using this operation.

In Section 2.4, I introduced Locality Sensitive Hashing (LSH) a seminal early method for solving the ANN search decision problems given in Definitions 2.3.1-2.3.2. As I discussed in Section 2.4.1, LSH for the inner product similarity samples hyperplanes uniformly from the unit sphere, relying on an asymptotic guarantee that as the number of hyperplanes increases the Hamming distance between the hashcodes will

Method	**Dependency**	**Learning Paradigm**	**Hash Function**	**Training Complexity**	**Properties**	**Section**
LSH	Independent	Unsupervised	$sgn(\mathbf{w}_k^{\intercal}\mathbf{x}+t_k)$	$O(KD)$	E_2	2.4.1
SKLSH	Independent	Unsupervised	$sgn(cos(\mathbf{w}_k^{\intercal}\mathbf{x}+t_k)+t_{k'})$	$O(KD)$	E_2	2.6.2.1
PCAH	Dependent	Unsupervised	$sgn(\mathbf{w}_k^{\intercal}\mathbf{x}+t_k)$	$O(min(N_{trd}^2 D, N_{trd}D^2))$	E_2,E_3,E_4	2.6.3.1
AGH	Dependent	Unsupervised	$sgn(\mathbf{w}_k^{\intercal}\mathbf{z}+t_k)$	$O(N_{trd}CK)$	E_1,E_2,E_3,E_4	2.6.3.4
ITQ	Dependent	Unsupervised	$sgn(\mathbf{RW}^{\intercal}\mathbf{x})$	$O(K^3)$	E_1,E_2,E_3,E_4	2.6.3.3
SH	Dependent	Unsupervised	$sgn(sin(\frac{\pi}{2}+j\pi(\mathbf{w}_k^{\intercal}\mathbf{x})))$	$O(min(N_{trd}^2 D, N_{trd}D^2))$	E_1,E_2,E_3,E_4	2.6.3.2
STH	Dependent	Supervised	$sgn(\mathbf{w}_k^{\intercal}\mathbf{x}+t_k)$	$O(N_{trd}DK+MN_{trd}^2K)$	E_1,E_2,E_3,E_4	2.6.4.4
KSH	Dependent	Supervised	$sgn(\mathbf{w}_k^{\intercal}\kappa(\mathbf{x})+t_k)$	$O(N_{trd}CK+N_{trd}^2CK+N_{trd}C^2K+C^3K)$	E_1,E_2	2.6.4.3
BRE	Dependent	Supervised	$sgn(\mathbf{w}_k^{\intercal}\kappa(\mathbf{x})+t_k)$	$O(KN_{trd}^2+KN_{trd}\log N_{trd})$	E_1,E_2	2.6.4.2
CVH	Dependent	Supervised	$sgn(\mathbf{w}_k^{\intercal}\mathbf{x}+t_k^x)$, $sgn(\mathbf{u}_k^{\intercal}\mathbf{z}+t_k^z)$	$O(N_{trd}D^2+D^3)$, D=$max(D_x,D_z)$	E_2,E_3,E_4	2.6.5.1
CMSSH	Dependent	Supervised	$sgn(\mathbf{w}_k^{\intercal}\mathbf{x}+t_k^x)$, $sgn(\mathbf{u}_k^{\intercal}\mathbf{z}+t_k^z)$	$O(KMN_{trd}D)$, D=$max(D_x,D_z)$	E_1,E_2	2.6.5.3
CRH	Dependent	Supervised	$sgn(\mathbf{w}_k^{\intercal}\mathbf{x}+t_k^x)$, $sgn(\mathbf{u}_k^{\intercal}\mathbf{z}+t_k^z)$	$O(D_x^2D_z+D_xD_z^2+D_z^3)$	E_1,E_2	2.6.5.2
PDH	Dependent	Supervised	$sgn(\mathbf{w}_k^{\intercal}\mathbf{x}+t_k^x)$, $sgn(\mathbf{u}_k^{\intercal}\mathbf{z}+t_k^z)$	$O(MN_{trd}^2K)$	E_1,E_2,E_4	2.6.5.4
IMH	Dependent	Supervised	$sgn(\mathbf{w}_k^{\intercal}\mathbf{x}+t_k^x)$, $sgn(\mathbf{u}_k^{\intercal}\mathbf{z}+t_k^z)$	$O(N_{trd}^3)$	E_1,E_2,E_3,E_4	2.6.5.5

Table 2.2: Categorisation of existing projection learning algorithms. This table is inspired in part by the categorisation given in Wang et al. (2010a). See Section 2.6 for detail on the specifics of each hash function. The last five hash functions are cross-modal. N is the total number of data-points, K is the hashcode length, C are a set of anchor data-points ($C \ll N_{trd}$), N_{trd} are the number of training data-points ($C < N_{trd} \ll N$), M are the number of iterations. Typically $N_{trd} = 1000\text{-}2000$, $K = 32\text{-}128$ and $C = 300$.

reflect the cosine similarity between any two data-points[8]. Nevertheless, as I pointed out in Section 2.4, randomly sampled LSH hyperplanes tend to lack discrimination and run a high risk of partitioning regions of the input feature space dense in related data-points. In practice this means that many hyperplanes (bits) and many hash tables are required for adequate retrieval effectiveness. Unfortunately, longer hashcodes and more hashtables require a greater main memory allocation for the LSH deployment. Recently researchers have turned to the question of how best to generate more compact and discriminative hashcodes by learning hyperplanes adapted to the distribution of the data (Liu et al. (2011, 2012); Weiss et al. (2008); Gong and Lazebnik (2011); Raginsky and Lazebnik (2009); Kulis and Darrell (2009); Zhang et al. (2010b)). It is these methods that form the focus in this part of the literature review.

Existing work on projection methods for hashing-based ANN can usefully be divided into *three* sub-fields based on the degree to which the distribution of the data informs the construction of the hashing hyperplanes: *data-independent* (Section 2.6.2), *data-dependent but unsupervised* (Section 2.6.3) and *data-dependent and supervised* (Sections 2.6.4-2.6.5). The projection methods I examine in this section are categorised in Table 2.2. I segment the field into these three areas and review related work under each category in Sections 2.6.2-2.6.5. The review will take us on a journey across a wide array of truly diverse techniques for generating hash functions, from random projections, kernel functions, spectral methods to boosting. I attempt to be as thorough as possible in our coverage of existing related work. Nevertheless, the literature on projection is truly vast due to its popularity as a research topic and therefore it will be impossible to provide an exhaustive coverage here due to space constraints. Instead I focus in detail on the more well-known models across each category whose authors have made the codebase freely available to the research community. Both of these points ensure that any claims I make in this thesis are both meaningful (e.g. stemming from results collected on the same experimental framework and on the same dataset splits) and based upon results from competitive baselines. I point the interested reader to two recently published review articles of Wang et al. (2014) and Grauman and Fergus (2013) for an additional overview of this part of the field. Note further that all of the hashing models I review restrict themselves to search over a single hash table ($L = 1$) as is the tradition in the literature. Methods that explicitly *learn* multiple hashtables in a data-dependent manner are an interesting sub-field but are out of the

[8]Goemans and Williamson (1995) showed that the expected Hamming distance between two bit vectors formed by hash functions sampled from $\mathcal{H}_{cosine}$ will approximate the angle between the vectorial feature representation of the corresponding data-points in the input feature space.

scope of this review. The reader is encouraged to see Xu et al. (2011) and Liu et al. (2013) for research in this direction.

2.6.1 The Four Properties of an Effective Hashcode

Before I discuss individual models for projection, I will firstly examine several properties that contribute to making an effective hashcode for nearest neighbour search. The seminal work on Spectral Hashing (SH) by Weiss et al. (2008) first codified four properties of an effective hashcode (E_1-E_4):

- E_1: The hashcode should have *low Hamming distance* to the hashcodes of similar data-points.
- E_2: The hashcode should be *efficiently computable* for a novel query data-point.
- E_3: The bits of the hashcode should have *equal probability* of being 0 or 1.
- E_4: The different bits of the hashcode should be *pairwise independent*.

While I have previously discussed the importance of the first property (E_1) in the context of LSH (Section 2.4) and binary quantisation (Section 2.5), I have so far not discussed the remaining criteria (E_2-E_4). The second property (E_2) is crucial for applying a hashing scheme in practice. Given a novel data-point we should be able to rapidly compute its hashcode so that the overall query time is kept to a minimum. This is known in the learning to hash literature as *out-of-sample extension*. LSH has a straightforward and computationally efficient method for out-of-sample-extension: simply multiply the query data-point by the matrix where each column constitutes the normal vector of a randomly sampled hyperplane followed by sign thresholding (Section 2.4). The last two properties target the *efficiency* E_3 and *compactness* E_4 of the hashcode. Property E_3 requires each hyperplane to generate a balanced partition of the data by splitting the dataset into two partitions of equal size i.e. $\sum_{i=1}^{N_{trd}} h_k(\mathbf{x}_i) = 0$. By the principle of maximum entropy this will maximise the information captured by the associated bit (Baluja and Covell (2008)). This constraint has the desirable effect of mapping an equivalent number of data-points to each hashcode and therefore balancing the occupancy of the hashtable buckets[9]. At query time we therefore avoid the degenerate case of having to examine an unnecessarily large number of nearest neighbours

[9]Wang et al. (2012) showed how the NP-hard property E_3 could be relaxed (and therefore implemented) by showing that it is equivalent to maximising the variance for the kth bit. Enforcing property E_3 might be sub-optimal, however, if it causes a cluster of related data-points to be partitioned into separate buckets. Usually such a situation can be remedied by using multiple independent hashtables.

in a given hashtable bucket. The fourth property E_4 targets hashcode compactness by eliminating any redundant bits that capture the same information on the input feature space. Ideally any hashing scheme should seek to minimise the number of bits in the hashcode to conserve storage and computation time. The vast majority of the data-dependent projection schemes introduced since the seminal work of Weiss et al. (2008) attempt to learn hashing hyperplanes that generate hashcodes with as many of these four properties as possible. I will study the extent to which these properties can be simultaneously preserved during the optimisation of the hashing hyperplanes in Sections 2.6.2-2.6.4.

2.6.2 Data-Independent Projection Methods

Aside from Locality Sensitive Hashing (LSH) which I reviewed in detail in Section 2.4, I will discuss one other data-independent hashing method in this thesis, Locality Sensitive Hashing from Shift Invariant Kernels (SKLSH) (Raginsky and Lazebnik (2009)) that extends LSH to the preservation of kernel similarity (Section 2.6.2.1).

2.6.2.1 Locality Senstive Hashing from Shift Invariant Kernels (SKLSH)

Locality Sensitive Hashing from Shift Invariant Kernels (SKLSH) extends LSH to the preservation of similarity between data-points as defined by an appropriate kernel function $\left\{\kappa : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}\right\}$ such as the Gaussian kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \|\mathbf{x}i - \mathbf{x}_j\|^2/2)$ or the Laplacian Kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|_1/2)$ where $\gamma \in \mathbb{R}$ is the kernel bandwidth parameter. In essence the method is similar to LSH but with a different definition of the hash function family $\mathcal{H}$ due to the different similarity preservation required. The crux of this hashing model is to construct an embedding $\left\{g : \mathbb{R}^D \rightarrow \{0,1\}^K\right\}$ such that if two data-points are similar as defined by the kernel function i.e. $\kappa(\mathbf{x}_i, \mathbf{x}_j) \approx 1$ then there will be a high degree of overlap between their hashcodes i.e. $d_{hamming}(g(\mathbf{x}_i), g(\mathbf{x}_j)) \approx 0$, and vice-versa for the situation when $\kappa(\mathbf{x}_i, \mathbf{x}_j) \approx 0$. To construct a mapping with this property Raginsky and Lazebnik (2009) formulate a low-dimensional projection function given by $\Psi^K : \mathbb{R}^D \rightarrow \mathbb{R}^K$. This projection uses the random Fourier features of Rahimi and Recht (2007) that provide a guarantee that the inner product between the two transformed data-points approximates the output of a *shift invariant* kernel[10] $\Psi_k(\mathbf{x}_i) \cdot \Psi_k(\mathbf{x}_j) \approx \hat{\kappa}(\mathbf{x}_i - \mathbf{x}_j)$. The random Fourier features mapping is given in Equation 2.22

[10]A shift invariant kernel is defined as: $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \hat{\kappa}(\mathbf{x}_i - \mathbf{x}_j)$.

$$\Psi_k(\mathbf{x}_i) = \sqrt{2}cos(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k) \tag{2.22}$$

where for the Gaussian kernel $\mathbf{w}_k \sim \mathcal{N}(0, \gamma \mathbf{I}_{D\times D})$ and $t_k \sim Unif[0, 2\pi]$. The contribution of Raginsky and Lazebnik (2009) is to use this embedding as the centerpiece of a novel hash function (Equation 2.23)

$$h_k(\mathbf{x}_i) = \frac{1}{2}[1 + sgn(cos(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k) + t_{k'})] \tag{2.23}$$

where sgn denotes the sign function adjusted so that $sgn(0) = -1$ and $t_{k'} \sim Unif[-1, 1]$. Raginsky and Lazebnik (2009) provide a proof that hashing the data-points with K randomly sampled hash functions will yield a binary embedding whose Hamming distance approximates the desired shift invariant kernel similarity. As the hyperplanes are sampled randomly the training time complexity of this algorithm is a low $O(DK)$. SKLSH satisfies property E_2 of an effective hashcode, namely efficient computation of hashcodes.

2.6.3 Data-Dependent (Unsupervised) Projection Methods

In this section I will provide a critical appraisal of relevant related work that learns the hashing hyperplanes in a data-dependent manner but without the need for supervisory information in the form of user provided pairwise constraints on data-point similarity or class labels. All of the unsupervised data-dependent hashing models I review in this section learn the hashing hyperplanes by formulating a *trace minimisation/maximisation* problem which is solved in closed form as an eigenvalue problem or using singular value decomposition (SVD). These hashing methods rely directly on well established methods of linear and non-linear dimensionality reduction, specifically Principal Components Analysis (PCA) and Laplacian Eigenmaps (LapEig). Given the widespread use of matrix factorisation in the learning to hash literature, including many methods I do not review here, I give a brief introduction to this important solution strategy before I review the individual hashing algorithms themselves in Section 2.6.3.1-2.6.3.4[11].

There are effectively two main strategies for performing a dimensionality reduction on a dataset $\mathbf{X} \in \mathbb{R}^{N\times D}$ to obtain a new dataset $\mathbf{Y} \in \mathbb{R}^{N\times K}$ where $K \ll D$. The first method involves finding an explicit linear transformation of the data characterised by

[11]For more detail on trace optimisation and eigenproblems for dimensionality reduction the reader is pointed to the excellent article of Kokiopoulou et al. (2011).

a projection matrix $\mathbf{W} \in \mathbb{R}^{D \times K}$ into the lower dimensional space ($\mathbf{Y} = \mathbf{XW}$). PCA is a well-known member of this *projective* category. The second method computes a non-linear low-dimensional embedding $\mathbf{Y} \in \mathbb{R}^{N \times K}$ directly, without first finding an explicit mapping function. These latter methods, of which LapEig is a prime example, typically impose neighbourhood constraints such that close by data-points in the original space are close-by in the reduced space. Despite these differences, both categories can be neatly unified by a standard trace maximisation objective function (Equation 2.24)

$$\begin{aligned} \operatorname{argmax}_{\mathbf{V} \in \mathbb{R}^{N \times K}} \quad & tr(\mathbf{V}^\intercal \mathbf{A} \mathbf{V}) \\ \text{subject to } & \mathbf{V}^\intercal \mathbf{1} = 0 \\ & \mathbf{V}^\intercal \mathbf{B} \mathbf{V} = \mathbf{I}^{K \times K} \end{aligned} \tag{2.24}$$

where $\mathbf{A}$ is a symmetric matrix, $\mathbf{B}$ is positive definite matrix, $\mathbf{V}$ is an orthonormal[12] matrix and $tr(A) = \sum_i A_{ii}$. The exact specification of these matrices is projection function dependent. I will concretely define $\mathbf{A}$, $\mathbf{B}$, $\mathbf{V}$ including their dimensionalities in Sections 2.6.3.1-2.6.3.4. But as a way of proving an immediate intuitive example, in the context of Principal Component Analysis (PCA) we have $\mathbf{A} = \mathbf{X}^\intercal \mathbf{X}$, $\mathbf{V} = \mathbf{W} \in \mathbb{R}^{D \times K}$ and $\mathbf{B} = \mathbf{I} \in \mathbb{R}^{D \times D}$. Therefore maximising the trace (Equation 2.24) in this case is equivalent to finding the principal directions in the data that capture the maximum variance in the input feature space.

The trace maximisation in Equation 2.24 can be solved as a general eigenvalue problem $\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{B}\mathbf{v}_i$, where $\mathbf{v}_i$ is the i^{th} eigenvector with eigenvalue λ_i (Saad (2011), Kokiopoulou et al. (2011)). This part of the learning to hash literature can now be distilled to its essence: in order to learn a set of data-dependent hash functions we shape our desired hashing optimisation problem into a form that resembles this template (Equation 2.24) and then we can simply solve for the K eigenvectors of a standard eigenvalue problem. This particular optimisation problem is easily solved using off the shelf solvers such as `eigs` or `svd` in Matlab. The main work in deriving an unsupervised data-dependent hash function can be summarised with the following standard four-step procedure:

1. Manipulating the problem into a matrix trace minimisation/maximisation (Equation 2.24).

[12] An orthonormal matrix $\mathbf{V}$ is a square matrix with real values whose columns and rows are orthogonal unit vectors. That is, $\mathbf{V}$ has the property $\mathbf{V}^\intercal \mathbf{V} = \mathbf{V}\mathbf{V}^\intercal = \mathbf{I}$, where $\mathbf{I}$ denotes the identity matrix.

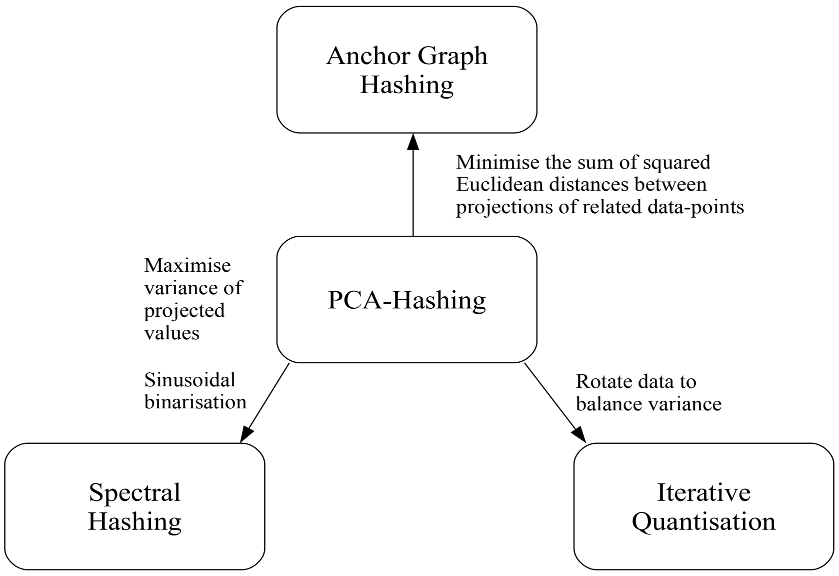

Anchor Graph Hashing
Minimise the sum of squared Euclidean distances between projections of related data-points
Maximise variance of projected values
PCA-Hashing
Sinusoidal binarisation
Rotate data to balance variance
Spectral Hashing
Iterative Quantisation

Figure 2.14: The data-dependent (unsupervised) hashing models are intimately related. This diagram illustrates one interpretation of that relationship where PCA is very much the centerpiece. The directed arcs are labelled with the operation necessary to transform one model into another model pointed to by the arc. See Section 2.6.3 for a full description of the four models.

2. Solving the optimisation objective as an eigenvalue problem or by performing a SVD. The K eigenvectors or right-singular vectors are the normal vectors of the hashing hyperplanes.

3. Dealing with the imbalanced variance resulting from the matrix factorisation.

4. Construct an out-of-sample extension in the case of a non-projective mapping.

We will see these four design principles in all four data-dependent hashing methods I review in this section. Specifically I will review PCA hashing (PCAH) (Section 2.6.3.1), Spectral Hashing (SH) (Section 2.6.3.2), Iterative Quantisation (ITQ) (Section 2.6.3.3) and Anchor Graph Hashing (AGH) (Section 2.6.3.4). In three out of four of the methods PCA extracts the directions of maximum variance which are then used as the hashing hyperplanes (PCAH, SH, ITQ). The contributions of the majority of these approaches lie in Step 3 where a sensible strategy is sought for minimising the impact of the imbalanced variance across hyperplanes, a phenomenon that reduces the quality of the hashcodes from lower principal components. The final method, AGH, takes a different tact (Section 2.6.3.4) and computes an eigenfunction extension of graph Laplacian eigenvectors, largely basing the hashcode learning on the Laplacian

Eigenmap dimensionality reduction algorithm. The objective of all of the presented algorithms is to learn K hash functions $\left\{h_k : \mathbb{R}^D \rightarrow \{0,1\}\right\}_{k=1}^{K}$ that can be concatenated to generate hashcodes for unseen data-points.

I present a diagram summarising one interpretation of the relationship between these hashing algorithms in Figure 2.14.

2.6.3.1 Principal Components Analysis Hashing (PCAH)

Principal components analysis (PCA) (Hotelling (1933)) has proven to be by far the most popular low dimensional embedding for data-dependent hashing schemes, with a large body of seminal works manipulating a PCA embedding to achieve superior retrieval accuracy over unsupervised hashing schemes (Kong and Li (2012b); Gong and Lazebnik (2011); Weiss et al. (2008); Wang et al. (2012)). I will therefore begin the review by examining the most basic instantiation of a PCA-based hashing scheme: namely computing the principal directions of the data and using the singular vectors with the highest singular values directly as the hashing hyperplanes without any further modification (Wang et al. (2010b)).

I assume without any loss of generality that the training data in $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D}$ has been centred by subtracting off the mean i.e. $\sum_{i=1}^{N_{trd}} \mathbf{x}_i = 0$. The standard maximum variance PCA objective can then be stated as in Equation 2.25

$$\begin{aligned} \operatorname{argmax}_{\{\mathbf{w}_k \in \mathbb{R}^D\}_{k=1}^{K}} \quad & \frac{1}{N_{trd}} \sum_k \mathbf{w}_k^\intercal \mathbf{X}^\intercal \mathbf{X} \mathbf{w}_k \\ = \quad & \frac{1}{N_{trd}} tr(\mathbf{W}^\intercal \mathbf{X}^\intercal \mathbf{X} \mathbf{W}) \\ \text{subject to } & \mathbf{W}^\intercal \mathbf{W} = \mathbf{I} \end{aligned} \tag{2.25}$$

where $tr(\mathbf{A}) = \sum_i A_{ii}$ denotes the matrix trace operator and $\mathbf{W} \in \mathbb{R}^{D \times K}$ is the matrix with columns $\mathbf{w}_k$. The constraint $\mathbf{W}^\intercal \mathbf{W} = \mathbf{I}$ requires the learnt hyperplanes to be pairwise orthogonal which can be thought of as a relaxed version of the pairwise independence property for bits (property E_4 in Section 2.6.1). Equation 2.25 is identical to Equation 2.24 with $\mathbf{A} = \mathbf{X}^\intercal \mathbf{X}$, $\mathbf{V} = \mathbf{W}$ and $\mathbf{B} = \mathbf{I}$. Therefore the $\left\{\mathbf{w}_k \in \mathbb{R}^D\right\}_{k=1}^{K}$ maximising Equation 2.25 are exactly the right singular vectors with the largest singular values which can be obtained using SVD on $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D}$ in $O(min(N_{trd}^2 D, N_{trd} D^2))$ operations. The PCA solution $\mathbf{W} \in \mathbb{R}^{D \times K}$, where each column constitutes a principal component, can be interpreted as a rigid rotation of the feature space such that each succeeding coordinate captures as much of the variance of the input data as possible.

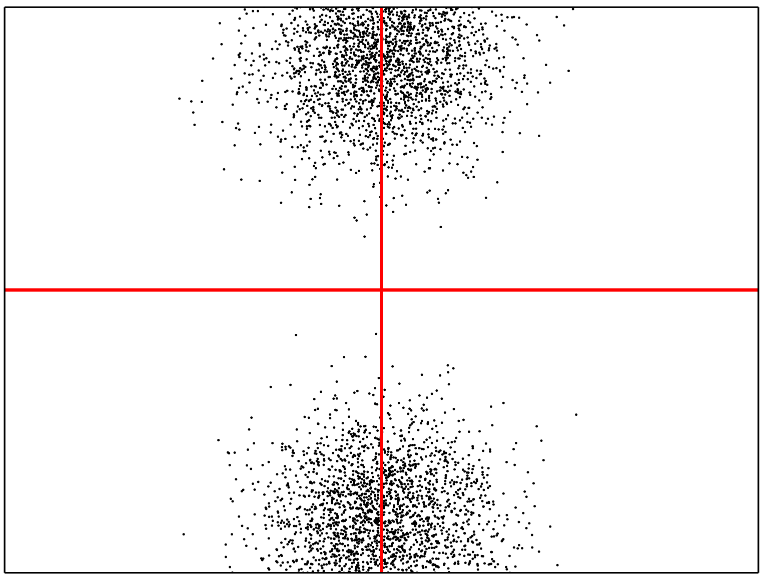


Figure 2.15: Plot displaying the PCA principal components $\mathbf{w}_1 \in \mathbb{R}^D$, $\mathbf{w}_2 \in \mathbb{R}^D$ (shown as perpendicular lines) for the data-points indicated by the black dots. Data has been aligned to the principal axes. The two principal components point in the directions of greatest variance of the data. These components are used as the vectors normal to the hashing hyperplanes in the PCA hashing (PCAH) algorithm.

For a K-bit hashcode it is common to take the K right-singular vectors with the highest singular values as the hashing hyperplanes while t_k is set to zero given that the data is mean-centered. The PCAH hash function is given in Equation 2.26.

$$h_k(\mathbf{x}_i) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{x}_i)) \tag{2.26}$$

Using PCA to generate hash functions can be thought of as attaining properties E_2, E_3, E_4 of an effective hashcode as identified in Section 2.6.1.

While the use of PCA is popular within the learning to hash literature, I mention here a number of disadvantages with using this matrix factorisation for generating hashcodes. Firstly, SVD is computationally expensive making this approach generally unattractive for databases with a large number of data-points and/or of a high dimensionality. Secondly, the number of bits K can never be greater than the dimensionality of the dataset D. In a practical hashing deployment, we first generate a very long hashcode for a data-point and then divide the hashcode up into L segments each of which provide the indices into the buckets of L hashtables. The fact that we must always have $K \leq D$ means that PCAH effectively places a restrictive upper bound on the number of hashtables L and hashcode lengths K we can use with this method. Finally, the singular vectors with the lowest singular values are likely to be unreliable, capturing little variance in the feature space. Using these singular vectors as hyperplane normal vectors in Equation 2.26 is likely to result in poor quality hashcodes that do not discriminate well between data-points. This latter issue, which I term the *imbalanced variance problem*, resulted in a flurry of additional research that specifically examined how best to extract the most information from the singular vectors with the highest singular values (Sections 2.6.3.2, 2.6.3.4) or that transform the original data-space so that the learnt hyperplanes capture an equal amount of the variance (Section 2.6.3.3).

2.6.3.2 Spectral Hashing (SH)

Spectral Hashing (Weiss et al. (2008)) (SH) was one of the earliest proposed schemes for data-dependent hashing and can be seen as the spark that ignited interest in data-dependent hashing within the field of Computer Vision. SH provides a standard framework for graph-based hashing and is central to unsupervised and supervised hashing models proposed later in the learning to hash literature. I therefore spend some time in this section drilling into the fine details of the algorithm. As I discussed in Section 2.6.1, SH placed the requirements of an “effective hashcode” on a firm theoretical grounding by introducing four properties (E_1, E_2, E_3, E_4) that such hashcodes should exhibit. In contrast to simply binarising the projections onto the first K principal components as is done in PCAH (Section 2.6.3.1), a procedure which is unlikely to generate hashcodes with the desired properties, SH examines the extent to which we can integrate three of the properties E_1, E_3, E_4 directly into the optimisation problem as the objective function (E_1) and constraints (E_3, E_4). The optimisation problem introduced by SH is given in Equation 2.27

$$\begin{aligned}
\text{argmin}_{\mathbf{Y}\in\mathbb{R}^{N_{trd}\times K}} \quad & \sum_{ij} S_{ij}\|\mathbf{y}_i - \mathbf{y}_j\|^2 \\
= \quad & tr(\mathbf{Y}^\intercal(\mathbf{D}-\mathbf{S})\mathbf{Y}) \\
\text{subject to } \mathbf{Y} \in & \{-1,1\}^{N_{trd}\times K} \\
& \mathbf{Y}^\intercal\mathbf{1} = \mathbf{0} \\
& \mathbf{Y}^\intercal\mathbf{Y} = N_{trd}\mathbf{I}^{K\times K}
\end{aligned} \tag{2.27}$$

where $D_{ii} = \sum_j S_{ij}$ is the diagonal degree matrix of the adjacency matrix $\mathbf{S} \in \mathbb{R}^{N_{trd}\times N_{trd}}$. Weiss et al. (2008) assume that the Euclidean distance between the input data-points is to be preserved and therefore $S_{ij} = exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/\gamma^2)$ is an appropriate similarity, where $\gamma \in \mathbb{R}$ is the kernel bandwidth parameter.

This objective function seeks to learn hashcodes $\left\{\mathbf{y}_i \in \{-1,1\}^K\right\}_{i=1}^{N_{trd}}$ where the average Hamming distance between similar neighbours is minimised, while satisfying bit balance and bit independence constraints. The constraint $\mathbf{Y}^\intercal\mathbf{1} = \mathbf{0}$ codifies property E_3 in requiring the bits to form a balanced partition of the feature space while constraint $\mathbf{Y}^\intercal\mathbf{Y} = N_{trd}\mathbf{I}^{K\times K}$ seeks bits that are pairwise uncorrelated which approximates property E_4. Unfortunately this optimisation problem is NP-hard even for a single bit, which can be proved with a reduction to the balanced graph partitioning problem which is well known to be NP-hard[13]. In order to make the optimisation problem tractable Weiss et al. (2008) use the spectral relaxation trick (Shi and Malik (2000)) removing the integrality constraint and letting the projection matrix $\mathbf{Y} \in \mathbb{R}^{N_{trd}\times K}$ consist of real numbers (Equation 2.28).

$$\begin{aligned}\operatorname{argmin}_{\mathbf{Y}\in\mathbb{R}^{N_{trd}\times K}} \quad & tr(\mathbf{Y}^\intercal(\mathbf{D}-\mathbf{S})\mathbf{Y}) \\ \text{subject to } & \mathbf{Y} \in \mathbb{R}^{N_{trd}\times K} \\ & \mathbf{Y}^\intercal\mathbf{1} = \mathbf{0} \\ & \mathbf{Y}^\intercal\mathbf{Y} = N_{trd}\mathbf{I}^{K\times K}\end{aligned} \tag{2.28}$$

Equation 2.28 is identical to Equation 2.24 with $\mathbf{A} = \mathbf{D}-\mathbf{S}$ and $\mathbf{V} = \mathbf{Y}$. The solutions of Equation 2.28 are therefore the K eigenvectors with minimal eigenvalue of the graph Laplacian $\mathbf{D}-\mathbf{S}$[14]. The rows of the spectral embedding matrix $\mathbf{Y} \in \mathbb{R}^{N_{trd}\times K}$ can be interpreted as the coordinates of each data-point in the low-dimensional embedding. Solving Equation 2.28 ensures that data-points deemed close by the neighbourhood graph $\mathbf{S} \in \{0,1\}^{N_{trd}\times N_{trd}}$ are mapped nearby in the embedded space, preserving the local distances. The time complexity of solving Equation 2.28 is approximately $O(N_{trd}^2 K)$. Unfortunately, the graph Laplacian eigenvectors obtained in this way will only generate the hashcodes for the N_{trd} training data-points leaving open the question of out-of-sample extension. A common way of solving this problem in the context of spectral methods is to compute the Nyström extension (Bengio et al. (2004); Williams and Seeger (2001)). However without making a suitable approximation (see Section 2.6.3.4) this procedure is just as costly ($O(N_{trd}K)$) as performing a brute-force search through the database making it unattractive for encoding unseen data-points at query time. To circumvent this issue Weiss et al. (2008) make a simple approximation by

[13]The interested reader is pointed to Weiss et al. (2008) for a proof.
[14]The trivial eigenvector **1** with eigenvalue 0 is ignored.

assuming the projected data is sampled from a multi-dimensional uniform distribution. In doing so they show that an efficient out-of-sample extension can be obtained by simply computing the one-dimensional Laplacian eigenfunctions given by Equation 2.29.

$$\Psi_{kj}(y_i^k) = sin(\frac{\pi}{2} + \frac{f\pi}{b_k - a_k} y_i^k) \tag{2.29}$$

with eigenvalues given by Equation 2.30:

$$\lambda_{kf} = 1 - e^{-\frac{\gamma^2}{2}|\frac{f\pi}{b_k - a_k}|^2} \tag{2.30}$$

along the principal directions given by PCA, where $f \in \{1 \dots K\}$ is the frequency, a_k, b_k are parameters of a uniform distribution estimated for projected dimension k and $y_i^k \in \mathbb{R}$ denotes the projection of data-point $\mathbf{x}_i$ onto the k^{th} principal direction. For ease of exposition I split the SH algorithm into a training step in which the parameters of the uniform distribution approximation $\{a_k, b_k\}_{k=1}^K$ and the PCA principal directions $\{\mathbf{w}_k \in \mathbb{R}^D\}_{k=1}^K$ are estimated and an out-of-sample extension step in which the hash-codes of novel data-points are generated. Both steps are summarised in A and B below:

(A) Hash function training:

1. Extract K eigenvectors $\{\mathbf{w}_k \in \mathbb{R}^D\}_{k=1}^K$ by computing PCA on the training database $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D}$ and stack as the columns of matrix $\mathbf{W} \in \mathbb{R}^{D \times K}$.

2. Project $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D}$ onto the principal directions $\{\mathbf{w}_k \in \mathbb{R}^D\}_{k=1}^K$ by computing $\mathbf{Y} = \mathbf{XW}$ where $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$

3. Estimate a uniform distribution $(a_k, b_k)_{k=1}^K$ for each projected dimension by computing the maximum b_k and minimum a_k extent of each dimension where $a_k = min(\mathbf{y}^k)$, $b_k = max(\mathbf{y}^k)$

4. For each projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$ compute K analytical eigenfunctions $\{\Psi_{kf}\}_{f=1}^K$ and their associated eigenvalues $\{\lambda_{kf} \in \mathbb{R}\}_{f=1}^K$ given by Equations 2.29-2.30.

5. Sort the K^2 eigenvalues and select the K analytical eigenfunctions from $\{\bar{\Psi}_{kj}\}_{k,j=1}^K$ with the smallest overall eigenvalues. Denote these as $\{\bar{\Psi}_k\}_{k=1}^K$, their corresponding normal vectors as $\{\bar{\mathbf{w}}_k \in \mathbb{R}^D\}_{k=1}^K$ and the parameters of the associated

uniform distributions $\{\bar{a}_k, \bar{b}_k\}_{k=1}^{K}$. Retain all three sets for out-of-sample extension.

(B) Out-of-sample extension:

1. Compute the K-bit hashcode $g(\mathbf{q}) = [h_1(\mathbf{q}), h_2(\mathbf{q}), \ldots, h_K(\mathbf{q})]$ for query $\mathbf{q}$ with the K hash functions defined as in Equation 2.31 using $\{\bar{\Psi}_k, \bar{\mathbf{w}}_k, \bar{a}_k, \bar{b}_k\}_{k=1}^{K}$ retained in Step 5 of the pre-processing stage. Using Equation 2.29 in the hash function can be thought of as a sinusoidal partitioning to be contrasted with the cosine partitioning of the projected dimension of SKLSH (Section 2.6.2.1).

$$h_k(\mathbf{q}) = \frac{1}{2}(1 + sgn(\bar{\Psi}_k(\bar{\mathbf{w}}_k^{\intercal}\mathbf{q})) \tag{2.31}$$

The computational complexity of this algorithm is dominated by the $O(min(N_{trd}^2 D, N_{trd} D^2))$ operations required to perform PCA on the database. SH prefers to select directions that have a large spread $|b_k - a_k|$ and low spatial frequency f. For low-dimensional data ($D \approx K$) SH commonly chooses multiple sinusoidal eigenfunctions with gradually higher frequencies for those eigenvectors that are pointing in the directions of greatest variance. To see this, note that the greater the variance of a projected dimension $\mathbf{y}^k$ the greater the range of $|b_k - a_k|$ and the lower the value of the corresponding eigenvalue given by Equation 2.30. In low-dimensional settings SH therefore has the desirable property of assigning more bits to the directions of highest variance in the input space, effectively up weighting the contribution of more informative hyperplanes in the Hamming distance computation. This somewhat overcomes the issue of PCAH in which we are progressively forced to pick orthogonal directions that capture less and less of the variance in the input space. Front loading the bits onto the most informative hyperplanes is one way of overcoming the imbalanced variance problem (Section 2.6.3.1) and usually leads to a higher retrieval effectiveness (Liu et al. (2011); Moran et al. (2013b)). The effectiveness of this variable bit allocation across hashing hyperplanes provides an inspiration for my novel variable threshold quantisation algorithm outlined in Chapter 5. In high dimensional settings ($D \gg K$) where the top eigenvectors capture a similar degree of variance, SH degenerates into PCAH by selecting each PCA hyperplane only once.

Despite the higher retrieval effectiveness versus LSH reported in Weiss et al. (2008) the unrealistic assumption of a uniform distribution has proved to be a considerable limitation of this method. The Anchor Graph Hashing (AGH) algorithm of Liu et al. (2011) seeks to overcome this issue by making a clever approximation that permits an efficient application of the Nyström method for out-of-sample extension. I turn to AGH in Section 2.6.3.4.

2.6.3.3 Iterative Quantisation (ITQ)

While Spectral Hashing (SH) implicitly allocates more bits to the hyperplanes that capture a greater proportion of the variance in the input space in order to counteract the imbalanced variance problem, Iterative Quantisation (ITQ) seeks to balance the variance across PCA hyperplanes through a learnt rotation of the feature space. ITQ introduces an iterative scheme reminiscent of the k-means algorithm to find a rotation of the feature space $\mathbf{R} \in \mathbb{R}^{K \times K}$ so that the resulting projections onto the principal directions $\mathbf{W} \in \mathbb{R}^{D \times K}$ will minimise the quantisation error specified in matricial form in Equation 2.32

$$\begin{aligned} \text{argmin}_{\mathbf{B} \in \mathbb{R}^{N_{trd} \times K}, \mathbf{R} \in \mathbb{R}^{K \times K}} \quad & \|\mathbf{B} - \mathbf{YR}\|_F^2 \\ \text{where } & \mathbf{B} \in \{-1, 1\}^{N_{trd} \times K} \\ \text{subject to } & \mathbf{R}^\intercal \mathbf{R} = K\mathbf{I}^{K \times K} \end{aligned} \tag{2.32}$$

Equation 2.32 is similar to the orthogonal Procrustes[15] problem (Schönemann (1966)) in which we seek to transform one matrix into another using an orthogonal transformation matrix in such a way as to minimise the sum of the squares of the resulting residuals between the target matrix and the transformed matrix. In this case Equation 2.32 seeks a rotation matrix $\mathbf{R} \in \mathbb{R}^{K \times K}$ so that the squared Euclidean distance between the projection vectors $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$ and their associated binary vectors $\mathbf{B} = sgn(\mathbf{XW})$ is minimised, where PCA hyperplanes are stacked in the columns of $\mathbf{W}$. This optimisation is challenging as both matrices $\mathbf{B} \in \mathbb{R}^{N_{trd} \times K}$ and $\mathbf{R} \in \mathbb{R}^{K \times K}$ are initially unknown. To learn the optimal $\mathbf{R}$ we need to know optimal $\mathbf{B}$ and to learn the optimal $\mathbf{B}$ we need to know the optimal $\mathbf{R}$. This chicken and egg type problem can be solved with an iterative scheme akin to k-means that starts off with a random guess for $\mathbf{R}$, before refining the matrix through a two-step optimisation procedure in which both matrices

[15]For the interested reader this problem is named after a particular grisly Greek myth involving the protagonist Procrustes, a villain who offered unwitting travelers their much needed rest on a "magic" bed that could perfectly accommodate any visitor no matter their height. Unfortunately, Procrustes had a penchant for removing the arms and legs of his guests so that they could be perfectly accommodated on the bed.

Algorithm 4: ITERATIVE QUANTISATION (ITQ) (GONG AND LAZEBNIK (2011))

```
Input: Data-points $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D}$, PCA hyperplanes $\mathbf{W} \in \mathbb{R}^{D \times K}$, number of
       iterations $M$, randomly initialised rotation matrix $\mathbf{R} \in \mathbb{R}^{K \times K}$
Output: Optimised rotation matrix $\mathbf{R} \in \mathbb{R}^{K \times K}$
1 $\mathbf{Y} \leftarrow \mathbf{XW}$                      // Project data onto PCA hyperplanes
2 for $m \leftarrow 1$ to $M$ do
3     $\mathbf{B} \leftarrow sgn(\mathbf{YR})$              // Rotate data using $\mathbf{R}$ and quantise
4     $\mathbf{S\Omega\hat{S}}^\intercal \leftarrow SVD(\mathbf{B}^\intercal\mathbf{Y})$          // Perform SVD on $\mathbf{B}^\intercal\mathbf{Y}$
5     $\mathbf{R} \leftarrow \mathbf{\hat{S}S}^\intercal$             // Rotation minimising Eq 2.32 for fixed $\mathbf{B}$
6 end
7 return $\mathbf{R}$
```

are learnt individually with the other fixed (Gong and Lazebnik (2011)). The iterative ITQ algorithm is presented in Algorithm 4.

The key step in the ITQ algorithm is shown in Line 4 of Algorithm 4. In Hanson and Norris (1981) and Arun et al. (1987) it is shown that with a fixed target matrix $\mathbf{B}$ the sought after transformation $\mathbf{R}$ minimising the squared Euclidean distance can be obtained from the singular value decomposition (SVD) of matrix $\mathbf{B}^\intercal\mathbf{Y}$. With a fixed $\mathbf{R}$, Gong and Lazebnik (2011) show that the optimal $\mathbf{B}$ minimising Equation 2.32 can be obtained simply by using single bit quantisation (Section 2.5.1) (Line 3). In addition to properties E_1-E_2, ITQ approximately conserves properties E_3 and E_4 of an effective hashcode introduced in Section 2.6.1. The balanced partition property (E_3) is met by maximising the variance of the projections using PCA which was shown in Wang et al. (2010b) to be a good approximation to conserving E_3. E_4 is approximately met by computing PCA on the data as the resulting hyperplanes will be orthogonal, a relaxed version of the pairwise independence property. The most computationally expensive step of ITQ is in Line 4 where the SVD of a $K \times K$ matrix is computed. This step takes $O(K^3)$ operations, where K is the hashcode length. The learnt rotation matrix can then be used to construct an ITQ hashcode for an unseen query data-point $\mathbf{q} \in \mathbb{R}^D$ as given in Equation 2.33

$$g_l(\mathbf{q}) = \frac{1}{2}(1 + sgn(\mathbf{RW}^\intercal\mathbf{q})) \tag{2.33}$$

where I assume the data has been mean-centered so that the quantisation threshold

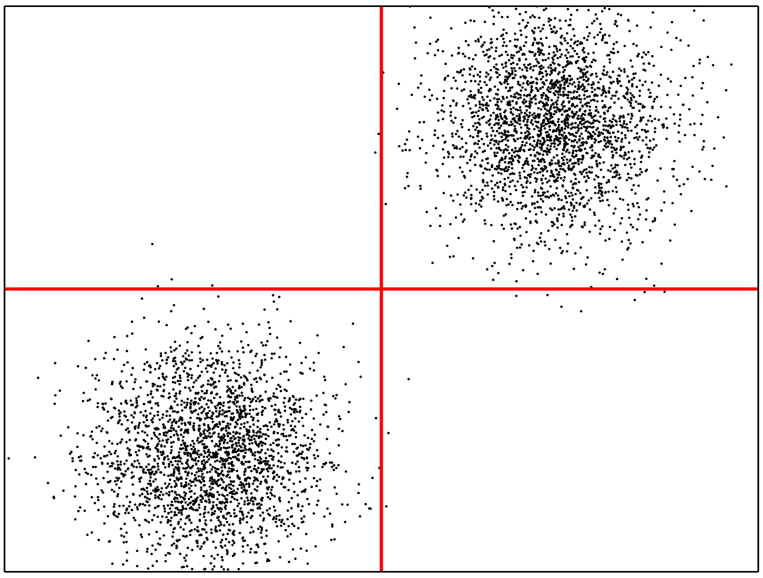


Figure 2.16: The effect of an ITQ rotation of the feature space. Here I show the same data as in Figure 2.15 but rotated by $\mathbf{R} \in \mathbb{R}^{K \times K}$ as found by ITQ over 100 iterations. The variance is more evenly distributed between the two hyperplanes (indicated as perpendicular lines) and the quantisation error is lower (no longer does a hyperplane directly cut through a cluster center). This is the optimisation objective of ITQ (Gong and Lazebnik (2011)).

$t_k = 0$.

I conclude with two personal observations on the ITQ algorithm. Firstly, such iterative two-step algorithms are a common and effective recipe for solving difficult optimisation problems within this field and crop up time and again in the literature. The need for a two-step algorithm is tied to the NP-hard problem of directly finding the optimal binary hashcodes. This issue can be tackled by making a continuous relaxation of $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$ as we first observed in the context of SH (Section 2.6.3.2). In this case a two-step procedure will find the best continuous approximation to the hashcodes followed by a second step that quantises the projections to generate the bits using either SBQ or one of the more sophisticated binarisation schemes introduced in Section 2.5. Being an approximation this process will produce sub-optimal hashcodes and so the challenge in most data-dependent projection models is to minimise the error in the continuous-to-binary conversion by learning the hashing hyperplanes in such a way that the resulting projections are more amenable to accurate binarisation. This algorithmic pattern is clearly evident in ITQ. Indeed, I will introduce my own novel data-dependent projection algorithm in Chapter 6 which has as its centerpiece a multi-step iterative algorithm that allows us to solve what would otherwise be a much more challenging optimisation problem. Only recently have authors turned to the more difficult problem of formulating data-dependent hashing algorithms that optimise for the binary hashcodes directly without making a continuous relaxation, see Section 2.6.4.2 and Liu et al. (2014) for an overview.

Secondly I comment briefly on why ITQ is considered a method of projection in this thesis, rather than quantisation. In Section 2.5, I defined a quantisation algorithm as one which learns one or more thresholds along a projected dimension that are then subsequently used in a thresholding operation to convert the real-valued projections to binary. ITQ is therefore not strictly a quantisation algorithm under the definition considered in this thesis as it does not directly convert real-valued projections to binary relying instead on SBQ (Section 2.5.1) for quantisation. I therefore categorise ITQ as a method for data-dependent projection as it works directly with the PCA hyperplanes rotating the data so that the resulting projections better preserve the locality structure of the input data-space.

2.6.3.4 Anchor Graph Hashing (AGH)

I previously described the Hierarchical Quantisation (HQ) algorithm employed by Anchor Graph Hashing (AGH) in Section 2.5.2. In this section I will focus exclusively on the AGH component that learns the projection function. AGH examines the same relaxed objective function as SH which I repeat in Equation 2.34 for reading convenience

$$
\begin{aligned}
\operatorname{argmin}_{\mathbf{Y}\in\mathbb{R}^{N_{trd}\times K}} \quad & tr(\mathbf{Y}^\intercal(\mathbf{D}-\mathbf{S})\mathbf{Y}) \\
\text{subject to } & \mathbf{Y}\in\mathbb{R}^{N_{trd}\times K} \\
& \mathbf{Y}^\intercal\mathbf{1}=\mathbf{0} \\
& \mathbf{Y}^\intercal\mathbf{Y}=N_{trd}\mathbf{I}^{K\times K}
\end{aligned}
\tag{2.34}
$$

The computational bottlenecks involved with this objective function are two-fold: firstly the similarity matrix $\mathbf{S}\in\mathbb{R}^{N_{trd}\times N_{trd}}$ requires $O(N_{trd}^2 D)$ computations to construct. Secondly as for any hashing method we need to compute the hashcodes for unseen query data-points using K hash functions $\{h_k:\mathbb{R}^D\rightarrow\{0,1\}\}_{k=1}^K$. Unfortunately solving Equation 2.34 and binarising the resulting eigenvectors will only provide the hashcodes for the training data-points used to construct the adjacency matrix

$\mathbf{S} \in \mathbb{R}^{N_{trd} \times N_{trd}}$. We need to extend the K graph Laplacian eigenvectors to K eigenfunctions $\{\Psi_k : \mathbb{R}^D \to \mathbb{R}\}_{k=1}^{K}$ which we can combine with an appropriate quantisation method to form the hash functions that will encode any data-point (seen or unseen). As mentioned in Section 2.6.3.2 this out-of-sample extension can be derived using the Nyström method (Bengio et al. (2004); Williams and Seeger (2001)) requiring $O(N_{trd}K)$ time for one data-point. Clearly this time complexity is not amenable to online hashcode generation for out-of-sample query data-points. The key takeaway message of the AGH algorithm is that a sparse, low-rank approximation of $\mathbf{S}$ can be implicitly manipulated through operations on a *truncated* similarity matrix $\mathbf{Z} \in \mathbb{R}^{N_{trd} \times C}$ $(C \ll N_{trd})$ known as the anchor graph. The approximate similarity matrix $\hat{\mathbf{S}} \in \mathbb{R}^{N_{trd} \times N_{trd}}$, which never needs to be explicitly computed, permits eigenfunction extension of the graph Laplacian in a time independent of the number of data-points while avoiding the need to manipulate the full dense similarity matrix $\mathbf{S}$. Furthermore, by computing the Nyström extension AGH is able to avoid the unrealistic separable uniform distribution assumption made by Spectral Hashing (described in Section 2.6.3.2).

More specifically the centerpiece of the AGH method is the concept of the *anchor graph* $\mathbf{Z} \in \mathbb{R}^{N_{trd} \times C}$, an approximation of a full data affinity graph, that only consists of the similarities from N_{trd} data-points to a small set of C anchors rather than the complete pairwise similarities between N_{trd}^2 data-points. These anchors are simply computed by running k-means over the training dataset and selecting the centroids $\{\mathbf{c}_i \in \mathbb{R}^D\}_{i=1}^{C}$ as the C anchor data-points. I first presented the anchor graph formulation in Equation 2.10 in the context of the Hierarchical Quantisation (HQ) method which for convenience I repeat in Equation 2.35

$$Z_{ij} = \begin{cases} \dfrac{exp(-d^2(\mathbf{x}_j, \mathbf{c}_i)/\gamma)}{\sum\limits_{i' \in \langle j \rangle} exp(-d^2(\mathbf{x}_j, \mathbf{c}_{i'}))/\gamma)} & \text{if } i \in \langle j \rangle \\ 0 & otherwise \end{cases} \tag{2.35}$$

where γ is the kernel bandwidth, $\{d(.,.) : \mathbb{R}^D \times \mathbb{R}^D \to [0,1]\}$ is a distance function and $\langle j \rangle \in \{1 \ldots R\}$ are the indices of the $R \ll C$ nearest anchors to $\mathbf{x}_j$ under the distance metric $d(.,.)$. As the number of anchors is much less than the number of data-points $(C \ll N_{trd})$, constructing the anchor graph is $O(N_{trd}CD)$ rather than $O(N_{trd}^2 D)$ for $\mathbf{S}$. Liu et al. (2011) show that the full similarity matrix $\hat{\mathbf{S}}$ can be approximated as

$\hat{\mathbf{S}} = \mathbf{Z}\Sigma^{-1}\mathbf{Z}^\intercal$ where $\Sigma = diag(\mathbf{Z}^\intercal\mathbf{1})$. The approximate similarity matrix $\hat{\mathbf{S}}$ has the computationally attractive properties of being sparse and low rank. The low rank property is exploited in the graph Laplacian eigenvector extraction by solving the eigenvalue system of the small $C \times C$ matrix $\Sigma^{1/2}\mathbf{Z}^\intercal\mathbf{Z}\Sigma^{-1/2}$. Given a bit budget of K in the hierarchical variant of their algorithm, Liu et al. (2011) select $K^{'} = K/2$ of the C eigenvectors with the highest eigenvalues as the hashing hyperplane normal vectors. Stacking the $K^{'}$ eigenvectors columnwise in matrix $\mathbf{V}$ in descending order of eigenvalue and the corresponding eigenvalues on the diagonal of matrix Λ, the required graph Laplacian eigenvectors $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K^{'}}$ can be computed as given in Equation 2.36.

$$\mathbf{Y} = \sqrt{N_{trd}}\mathbf{Z}\Sigma^{-1/2}\mathbf{V}\Lambda^{-1/2} = \mathbf{Z}\mathbf{W} \tag{2.36}$$

The training time complexity of computing $\mathbf{Y}$ is $O(N_{trd}CK^{'})$. The columns of the matrix $\mathbf{W} \in \mathbb{R}^{C \times K^{'}}$ can be seen as the normal vectors of $K^{'}$ hyperplanes partitioning the space $\mathbb{R}^C$ formed by the non-linear mapping in Equation 2.35. Liu et al. (2011) show that an out-of-sample extension can be achieved in two steps: firstly, the unseen query data-point $\mathbf{q}$ is non-linearly projected into the space $\mathbb{R}^C$ by computing the similarity of $\mathbf{q}$ to the C cluster centroids $\{\mathbf{c}_i \in \mathbb{R}^D\}_{i=1}^{C}$ using Equation 2.35. This operation results in a sparse transformed vector $\mathbf{z} \in \mathbb{R}^C$ which can also be interpreted as a kernelised feature map (Murphy (2012)). This step is subsequently followed by a linear projection of $\mathbf{z}$ onto the k-th hyperplane $\mathbf{w}_k \in \mathbb{R}^C$ partitioning the space $\mathbb{R}^C$. The AGH hash function is formed from both steps (Equation 2.37)

$$h_k(\mathbf{q}) = \frac{1}{2}(1 + sgn(\mathbf{w}_k^\intercal \mathbf{z})) \tag{2.37}$$

where I again assume the data is mean centered so that $t_k = 0$.

This hash function can be thought of as non-linearly mapping the data into a space where it is more likely to be linearly separable by linear decision boundaries. Given an unseen query data-point computing this out-of-sample extension takes $O(CD + CK^{'})$ operations, a testing time complexity that is a marked improvement over the $O(N_{trd}K)$ time complexity of the Nyström method[16]. Rather than generate one bit per hash function as suggested by Equation 2.37, the most accurate variant of AGH generates two bits for each of the resulting projected dimensions $\mathbf{y}^k = Y_{\bullet k}$ with $k \in [1, \dots, K^{'}]$. We

[16]Assuming $D \ll N_{trd}$, which is generally true for the most common image features such as Gist and SIFT.

previously discussed this Hierarchical Quantisation (HQ) algorithm in detail in Section 2.5.2.

2.6.3.5 A Brief Summary

In our first foray in data-dependent hashing algorithms I surveyed a selection of the more well-known unsupervised algorithms that position the hashing hyperplanes based on the distribution of the data. I reviewed Principal Components Analysis Hashing (PCAH) (Section 2.6.3.1), Spectral Hashing (SH) (Section 2.6.3.2) and Anchor Graph Hashing (AGH) (Section 2.6.3.4). I saw that all three models reviewed are closely related in their application of a well-known dimensionality reduction method, either Principal Components Analysis (PCA) or Laplacian Eigenmaps (LapEig), to learn the hashing hyperplanes.

Three out of four of the hashing models (PCAH, SH, ITQ) used PCA, setting the hashing hyperplanes to be the right singular vectors resulting from a SVD on the data matrix. Two of these models (SH, ITQ) highlighted the issue of *variance imbalance* in which the hyperplanes capturing a smaller amount of the variance are much less reliable for hashing. The upshot of this is that PCAH retrieval effectiveness declines markedly with longer hashcode lengths due to the incorporation of lower quality hyperplanes into the hashcode generation. To counter this degradation in performance SH assigns more hashcode bits to the hyperplanes with higher variance while ITQ rigidly rotates the feature space to explicitly balance the variance across hyperplanes. All three models show higher retrieval effectiveness than PCAH which assigns 1 bit per hyperplane or simply uses the PCA hyperplanes as is.

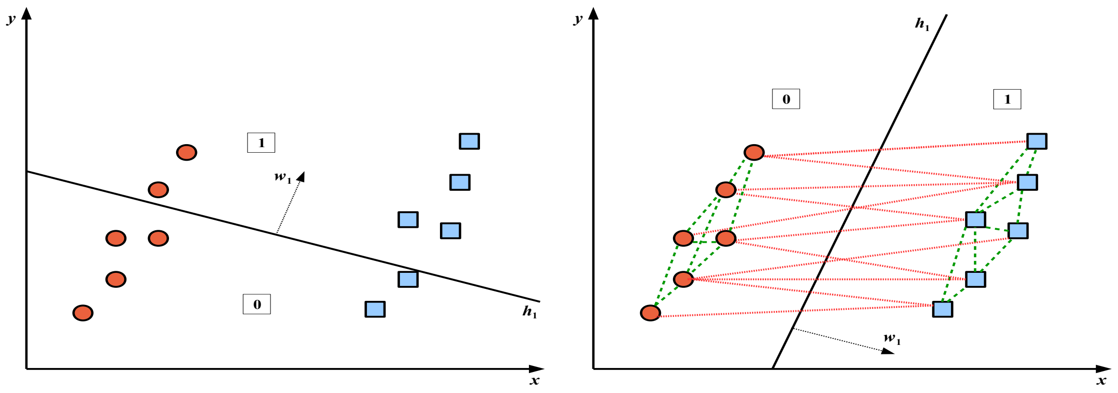
I also discussed how the AGH algorithm took a different strategy to the PCA-based hashing algorithms by using a LapEig-inspired dimensionality reduction. In this scenario a nearest neighbour graph was built from the input data which was then used in an eigenvalue problem to extract graph Laplacian eigenvectors. Given that LapEig is a non-projective dimensionality reduction these eigenvectors were shown to yield the hashcodes for only those data-points used in the neighbourhood graph computation. An appealing property of AGH is its computationally efficient method, based on the Nyström method of Williams and Seeger (2001), for out-of-sample extension to unseen data-points.

Aside from AGH which makes an honest attempt at reducing the computational complexity at training time, the downside with most of these hashing algorithms is the severe computational penalty $O(min(N^2D, ND^2))$ required for solving the SVD or eigenvalue problem making their application intractable for large-scale datasets of high dimensionality. Indeed, as we will see in forthcoming sections most data-dependent hashing models (both supervised and unsupervised) generally rely on a matrix factorisation.

2.6.4 Data-Dependent (Supervised) Projection Methods

In Section 2.4 and Sections 2.6.2-2.6.3 I reviewed a selection of state-of-the-art data-independent and data-dependent hashing models. The data-independent models preserve a similarity, such as the cosine or a kernel similarity, that is non data-adaptive and is therefore unlikely to do very well at capturing a user-defined notion of similarity across many different tasks. Moreover, the data-dependent (unsupervised) models assume, for example, that discriminative hashcodes can be generated from projected dimensions that capture the maximum variance in the input space. This relies on variance being a quantity that can effectively distinguish between unrelated data-points, an assumption which may not be valid in many datasets of practical interest, such as image datasets collated "in the wild" from the WWW that depict images of varying topic, quality and resolution. This is exacerbated by the well-known *semantic gap* problem in computer vision which highlights the gulf between the statistics of the images captured by low-level image features such as Gist and SIFT and the high-level semantic concepts that are depicted in the image (Smeulders et al. (2000)). A robust way of linking these two domains is one of the grand challenges in the sub-fields of object recognition and image annotation (Moran and Lavrenko (2015a)), and is also important in our selected task of image retrieval.

To mitigate the difficulties arising from the semantic gap and capture the complex relationships between data-points found in real-world datasets, such as whether two images depict a cat or a person, it is generally much better to learn a hash function from a small amount of available supervision in the form of human annotated class labels or pairwise cannot-link or must-link constraints that specify which data-point pairs should or should not have the same hashcodes. In the visual search domain, Grauman and Fergus (2013) highlight potential sources of supervisory information ranging from explicit labelling of a subset of the database, to known correspondences between points in image pairs and user feedback on image search results. It is this category of hashing model that I review in this section. In general, I define a supervised hashing model as a model that leverages the same type of information (e.g. class labels, metric distances)



(a) **Unsupervised** (b) **Supervised**

Figure 2.17: Supervised versus unsupervised projection function learning. Illustration of a situation where learning hashing hyperplanes based on pairwise user provided constraints can yield a more effective bucketing of the space than a partitioning based on maximum variance. Points with similar shapes and colours are 1-nearest neighbours. In Figure (a) hyperplane $\mathbf{h}_1 \in \mathbb{R}^D$ is learnt via PCA with its normal vector $\mathbf{w}_1 \in \mathbb{R}^D$ pointing in the direction of maximum variance in the data. Projecting data-points onto the normal vector $\mathbf{w}_1 \in \mathbb{R}^D$ places related data-points (indicated by the same shapes) into different buckets. In contrast Figure (b) illustrates the effect of constraining the hyperplane positioning by using a set of must-link (show as dotted lines) and cannot-link (shown as solid lines connecting the data-points) constraints. I only show a subset of the constraints for clarity. In this case all related data-points fall within the same bucket as each other yielding a more effective partitioning of the space.

in the hash function learning algorithm that was also used to compute the groundtruth information for evaluation purposes. In Figure 2.17, I illustrate a situation in which learning hyperplanes based on pairwise labels yields a more effective bucketing of the space than one based purely on captured variance.

In a similar manner to the data-dependent (unsupervised) models, I restrict my attention to a selection of the most well-known baselines from the literature and whose authors have made the codebase freely available to the research community thereby making a fair comparison to our own methods possible under identical experimental conditions. I review ITQ with a Canonical Correlation Analysis (CCA) embedding (ITQ + CCA) (Gong and Lazebnik (2011)), Supervised Hashing with Kernels (KSH) (Liu et al. (2012)), Binary Reconstructive Embedding (BRE) (Kulis and Darrell (2009)) and Self-Taught Hashing (STH) (Zhang et al. (2010b)). These four models

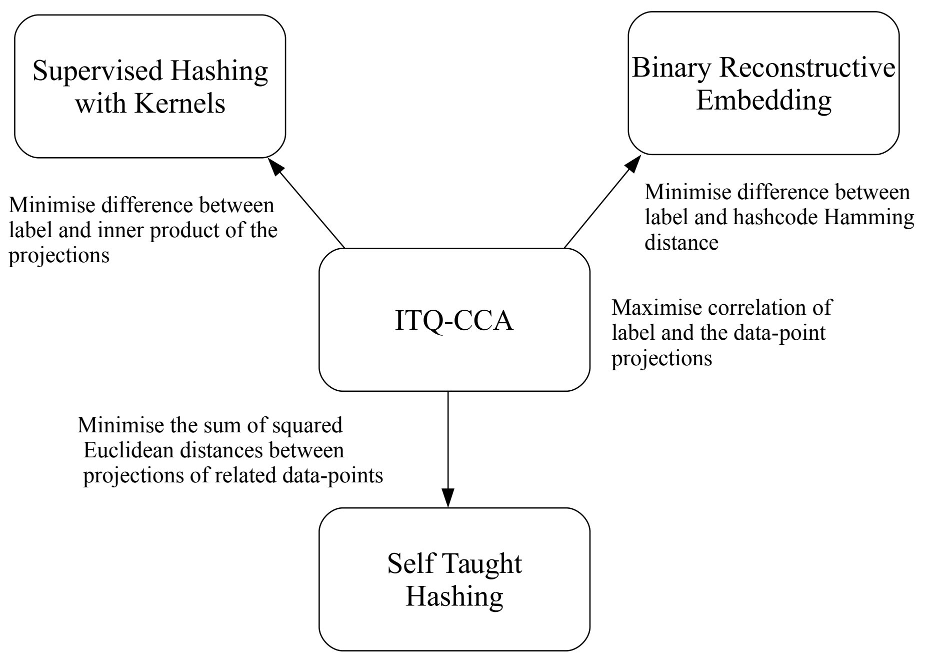

Supervised Hashing with Kernels
Binary Reconstructive Embedding
Minimise difference between label and inner product of the projections
Minimise difference between label and hashcode Hamming distance
ITQ-CCA
Maximise correlation of label and the data-point projections
Minimise the sum of squared Euclidean distances between projections of related data-points
Self Taught Hashing

Figure 2.18: Relationship between the four supervised hashing models reviewed in this section. The labels on the arrows indicate the transformation necessary to convert between the different models. The fundamental difference between the models arises in how the available labels are related to the projections/hashcodes so as to compute an error signal to adjust the hashing hyperplanes.

fundamentally differ only in how they use the available labels to derive an error signal that can then be used to adjust the positioning of the hashing hyperplanes. For example, BRE and KSH frame similar objective functions that attempt to minimise the difference between the labels and the hashcode distances (BRE and KSH). STH uses the LapEig objective which minimises the difference between the projections of data-points with the same label while ITQ+CCA frames an objective that maximises the correlation of the labels and data-point projections. The relationship between these four supervised hashing models is summarised in Figure 2.18.

2.6.4.1 ITQ + Canonical Correlation Analysis (CCA)

I reviewed the unsupervised variant of Iterative Quantisation (ITQ) in Section 2.6.3.3. ITQ learns an orthogonal rotation matrix $\mathbf{R} \in \mathbb{R}^{K \times K}$ that transforms PCA projected data in a way that minimises the error of mapping the data to the vertices of a binary hypercube. ITQ is independent of the method for generating the orthogonal hashing hyperplanes $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K]$ where $\mathbf{w}_k \in \mathbb{R}^D$, which in the case of the original algorithm was PCA. It is therefore straightforward to make ITQ into a supervised algorithm by using a supervised embedding to learn the hashing hyperplanes rather than PCA. Gong and Lazebnik (2011) replace PCA with Canonical Correlation Analysis (CCA) (Hardoon et al. (2003)) a well-known multi-view dimensionality reduction technique that explores the interaction between data vectors in two different feature spaces $\mathcal{X}$ and $\mathcal{Z}$. Assume we have N_{trd} training data-points in matrix $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D_x}$ and their associated labels in matrix $\mathbf{Z} \in \mathbb{R}^{N_{trd} \times D_z}$, where usually $D_x \neq D_z$. Each row of matrix $\mathbf{Z}$ is a binary indicator vector $\mathbf{z}_i \in \{0,1\}^{D_z}$ where a '1' indicates that the data-point $\mathbf{x}_i$ is tagged with that label and a '0' otherwise. The CCA algorithm finds two hyperplane normal vectors $\mathbf{w}_k \in \mathbb{R}^{D_x}$ and $\mathbf{u}_k \in \mathbb{R}^{D_z}$ so that the projections $\mathbf{X}\mathbf{w}_k$ and $\mathbf{Z}\mathbf{u}_k$ are maximally correlated (Equation 2.38).

$$\begin{aligned} &\operatorname{argmax}_{\mathbf{w}_k \in \mathbb{R}^D, \mathbf{u}_k \in \mathbb{R}^D} \quad \frac{\mathbf{w}_k \mathbf{X} \mathbf{Z} \mathbf{u}_k}{\sqrt{\mathbf{w}_k^\intercal \mathbf{X}^\intercal \mathbf{X} \mathbf{w}_k \mathbf{u}_k^\intercal \mathbf{Z}^\intercal \mathbf{Z} \mathbf{u}_k}} \\ &\text{subject to } \mathbf{w}_k^\intercal \mathbf{X}^\intercal \mathbf{X} \mathbf{w}_k = 1 \\ &\mathbf{u}_k^\intercal \mathbf{Z}^\intercal \mathbf{Z} \mathbf{u}_k = 1 \end{aligned} \tag{2.38}$$

This objective function can be maximised by solving the following generalised eigenvalue problem (Gong and Lazebnik (2011))

$$\mathbf{X}^\intercal \mathbf{Z} (\mathbf{Z}^\intercal \mathbf{Z} + \rho \mathbf{I})^{-1} \mathbf{Z}^\intercal \mathbf{X}^\intercal \mathbf{w}_k = \lambda_k^2 (\mathbf{X}^\intercal \mathbf{X} + \rho \mathbf{I}) \mathbf{w}_k \tag{2.39}$$

where λ_k is the eigenvalue and ρ is a regularisation constant set to 0.0001 in Gong and Lazebnik (2011). Repeatedly solving Equation 2.39 for directions that are orthogonal to all previously discovered hyperplanes gives K orthogonal hyperplanes with normals $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K]$ whose positioning in the feature space have been influenced by the supervisory signal. Having learnt the hyperplanes $\mathbf{W} \in \mathbb{R}^{D \times K}$ in modality $\mathcal{X}$ the remainder of the ITQ+CCA algorithm proceeds in the same way as for the unsupervised variant of ITQ (Section 2.6.3.3). If I denote $D = max(D_x, D_z)$, then the computational complexity of ITQ+CCA is bounded by $O(N_{trd}D^2 + D^3)$. This is made up of the $O(N_{trd}D^2)$ operations required to compute the covariance matrices and the $O(D^3)$ operations arising from the matrix multiplications, inversion and solving the eigenvalue problem (Rasiwasia et al. (2014)). Following a similar line of argument to ITQ, ITQ + CCA approximately preserves properties E_1-E_4 of an effective hashcode.

2.6.4.2 Binary Reconstructive Embedding (BRE)

Binary Reconstructive Embedding (BRE) is the only projection method I consider that does not make use of the spectral relaxation trick to circumvent the NP-hard optimisation problem of learning binary hashcodes directly. We were first introduced to this continuous relaxation in the context of Spectral Hashing (Section 2.6.3.2). Without making the spectral relaxation and dropping the sign function from the optimisation objective many approaches to data-dependent hashing are discontinuous and non-differentiable. The contribution of BRE is a novel optimisation objective and a coordinate descent algorithm that solves the discrete optimisation problem directly without appealing to a continuous relaxation. As we have seen before in this literature review many methods solve for a matrix $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$ of real-numbers and then binarise this matrix to reveal the hashcodes using, for example, single bit quantisation (SBQ). These two steps are disconnected and there is therefore no guarantee that the real-values in $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$ will reliably map to accurate binary hashcodes particularly if they are close to the threshold boundary (which is typically at zero for mean centered data). BRE brings both steps into the optimisation objective by retaining the sign function. I present the *supervised* variant of the BRE objective function in Equation 2.40

$$\begin{aligned}
&\text{argmin}_{\mathbf{W} \in \mathbb{R}^{D \times K}} \sum_{ij \in \mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}} \left\{ (1 - S_{ij}) - \frac{1}{K} \|g(\mathbf{x}_i) - g(\mathbf{x}_j)\|_2^2 \right\}^2 \\
&\quad \text{subject to } g(\mathbf{x}_i) = [h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \ldots, h_k(\mathbf{x}_i)]^\intercal \\
&\quad \text{where } h_k(\mathbf{x}_i) = \frac{1}{2} sgn(1 + \sum_{j=1}^{C} W_{jk} \kappa(\mathbf{x}_j, \mathbf{x}_i))
\end{aligned} \tag{2.40}$$

where $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ is an adjacency matrix with $S_{ij} = 1$ indicates $\mathbf{x}_i$ and $\mathbf{x}_j$ are related and 0 otherwise. N_{trd} data-points ($C < N_{trd} \ll N$) are sampled from the dataset to construct $\mathbf{S}$ and C data-points are sampled uniformly at random as the anchor points for efficient kernel computation. $\mathbf{W} \in \mathbb{R}^{C \times K}$ is initialised randomly, κ is a kernel function $\left\{\kappa : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}\right\}$. Kulis and Darrell (2009) set κ to be the linear kernel in the original publication.

The objective function in Equation 2.40 attempts to make the normalised Hamming distance low for those data-point pairs with $S_{ij} = 1$, and large otherwise. No part of this objective encourages the conservation of properties E_3 and E_4 of an effective hashcode.

In the case of both objective functions a kernelised feature map $\{\kappa : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}\}$ is computed against a small number C of randomly sampled data-points from the training dataset, and the mapped data projected onto a set of K hyperplane normal vectors $\{\mathbf{w}_k \in \mathbb{R}^C\}_{k=1}^K$. This formulation of the hash function is similar to that of Anchor Graph Hashing (Equation 2.37) except AGH maps the data non-linearly using an RBF kernel and uses k-means centroids as the C samples to construct the kernel. The retrieval effectiveness of BRE may benefit from a non-linear kernelised feature map although this formulation was not explored in the original publication.

Perhaps the most interesting contribution of BRE is the optimisation algorithm used to minimise Equation 2.40 with the sign function intact. To optimise the non-differentiable objective function Kulis and Darrell (2009) formulate a coordinate descent algorithm that cycles through each hash function one by one and finds the value minimising Equation 2.40 of a randomly chosen element W_{jk} of each hyperplane $\mathbf{W}_{\bullet k}$, while holding the remaining hyperplanes constant. Kulis and Darrell (2009) provide a closed form solution for computing the optimal W_{jk} in $O(N_{trd}^2)$ time. This procedure is repeated for the remaining hash functions. In total one iteration through all K hash functions takes $O(KN_{trd}^2 + KN_{trd} \log N_{trd})$ operations[17]. BRE meets properties E_1-E_2 of an effective hashcode. The hashcode bits generated by BRE are correlated (property E_3 is not conserved) given that the coordinate descent algorithm cycles through each hash function in turn updating the current hash function based on the optimised hyperplane normal vectors of previously examined hash functions. The benefit of tackling the discrete optimisation problem directly has recently garnered renewed attention in Liu et al. (2014) and Shen et al. (2015).

2.6.4.3 Supervised Hashing with Kernels (KSH)

Supervised Hashing with Kernels (KSH) formulates a kernelised hash function in a similar manner to AGH (Section 2.6.3.4) and BRE (Section 2.6.4.2) but proposes an entirely different and spectrally relaxed optimisation algorithm (Liu et al. (2012)). KSH exhibits the highest retrieval effectiveness compared to the supervised hashing models I discuss in this section and frequently appears in the literature as the de-facto baseline for comparison on the standard image datasets considered in this thesis. The familiar kernelised hash function is presented in Equation 2.41

[17]For ease of presentation I assume each of the N_{trd} training data-points forms N_{trd}-1 supervisory pairs with the other N_{trd}-1 training data-points in $\mathbf{S}$. In practice, for computational tractability, BRE randomly selects a much smaller sample of pairs (e.g. $0.05N_{trd}$) for each training datapoint.

$$h_k(\mathbf{q}) = sgn(\sum_{j=1}^{C} W_{jk}\kappa(\mathbf{x}_j,\mathbf{q}) + t_k) \tag{2.41}$$

where κ is the kernel function $\kappa : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$, $t_k \in \mathbb{R}$ is a scalar threshold and $\mathbf{W} \in \mathbb{R}^{C\times K}$ is a set of K hyperplane normal vectors. As for BRE and AGH a small number of C ($C \ll N$) data-points are sampled uniformly at random from the dataset $\mathbf{X} \in \mathbb{R}^{N\times D}$ to compute the required kernel similarities. In addition, N_{trd} data-points ($C < N_{trd} \ll N$) are sampled from the dataset to construct the adjacency matrix $\mathbf{S} \in \{-1,1\}^{N_{trd}\times N_{trd}}$, which acts as the training samples for learning the hash functions. The objective function of KSH (Equation 2.42) is very similar to the supervised BRE objective function, the only salient difference being the removal of the sign function and the computation of the inner product $(g^{\intercal}(\mathbf{x}_i)g(\mathbf{x}_j))$ between a pair of hashcodes for data-points $\mathbf{x}_i, \mathbf{x}_j$, rather than the Euclidean distance.

$$\begin{gathered}\text{argmin}_{\mathbf{W}\in\mathbb{R}^{C\times K}} \sum_{ij\in\mathbf{S}\in\mathbb{R}^{N_{trd}\times N_{trd}}} \left\{S_{ij} - \frac{1}{K}g^{\intercal}(\mathbf{x}_i)g(\mathbf{x}_j)\right\}^2 \\ \text{subject to } g(\mathbf{x}_i) = [h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), \dots, h_K(\mathbf{x}_i)]^{\intercal} \\ h_k(\mathbf{x}_i) = sgn(\sum_{j=1}^{C} W_{jk}\kappa(\mathbf{x}_j,\mathbf{x}_i))\end{gathered} \tag{2.42}$$

Recall from Section 2.6.4.2 that BRE retains the sign function and tackles the resulting NP-hard optimisation problem via a coordinate descent algorithm that measures the impact of flipping bits on the objective function value. In contrast KSH drops the sign function and performs the hashcode optimisation over a continuous space that admits a more efficient parameter update via gradient descent. KSH optimises each of the K hash functions sequentially by firstly initialising each hyperplane normal $\{\mathbf{w}_k \in \mathbb{R}^C\}_{k=1}^{K}$ by solving an eigenvalue problem which is then followed by a gradient descent optimisation to further refine the hyperplanes. To see how the KSH sequential optimisation algorithm works more clearly, I drop the sign function and rewrite Equation 2.42 to iterate over the K hash functions rather than data-point pairs (Equation 2.43)

$$\begin{gathered}\text{argmin}_{\mathbf{W}\in\mathbb{R}^{C\times K}} \sum_{k=1}^{K} \|K\mathbf{S} - \mathbf{y}^k(\mathbf{y}^k)^{\intercal}\|_F^2 \\ \text{where } y_i^k = \sum_{j=1}^{C} W_{jk}\kappa(\mathbf{x}_j,\mathbf{x}_i)\end{gathered} \tag{2.43}$$

where I have assumed that the data is mean centered, and therefore $t_k = 0$. Recall from Table A.1 in Appendix A that the notation $\mathbf{y}^k$ signifies the k^{th} *column* of the projection matrix $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$. It is possible to approximately solve Equation 2.43 by simply optimising each hyperplane individually giving K independent optimisation problems. Instead KSH opts for a solution strategy similar to that of BRE where the hyperplanes are solved in a sequential manner thereby instilling a degree of dependence between the hashcode bits. In the case of KSH this dependence is captured with a residue matrix $\mathbf{R} \in \mathbb{Z}_{+}^{N_{trd} \times N_{trd}}$ defined in Equation 2.44

$$\mathbf{R}^{k-1} = K\mathbf{S} - \sum_{l=1}^{k-1} \mathbf{y}^l (\mathbf{y}^l)^\intercal \tag{2.44}$$

The magnitude of $\mathbf{R}$ is related to the number of mismatches between the signs of data-point pairs where $S_{ij} = 1$ in the adjacency matrix. The higher the number of mismatches for a given data-point pair $(\mathbf{x}_i, \mathbf{x}_j)$ over the previous k-1 hash functions the greater the value of the corresponding element R_{ij}^{k-1} and the greater the influence that pair will have on learning of the k^{th} hash function. In this way the hash function learning is gradually biased towards correctly labelling those data-point pairs that were incorrectly labelled by hyperplanes learnt earlier in the optimisation procedure. Liu et al. (2012) show that the objective function in Equation 2.43 can be reduced to Equation 2.45

$$\begin{gathered} \operatorname{argmax}_{\mathbf{w}_k \in \mathbb{R}^C} \quad (\mathbf{K}\mathbf{w}_k)^\intercal \mathbf{R}^{k-1} (\mathbf{K}\mathbf{w}_k) \\ \text{where } K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to } (\mathbf{K}\mathbf{w}_k)^\intercal (\mathbf{K}\mathbf{w}_k) = L \end{gathered} \tag{2.45}$$

where $\mathbf{K} \in \mathbb{R}^{N_{trd} \times C}$ is the kernel matrix. Comparing the form of Equation 2.45 to the standard eigenvalue problem template presented in Equation 2.24 we can immediately see that the solution to this optimisation problem is the eigenvector with the largest eigenvalue of $\mathbf{K}^\intercal \mathbf{R}^{k-1} \mathbf{K}\mathbf{w}_k = \lambda \mathbf{K}^\intercal \mathbf{K}\mathbf{w}_k$. In the KSH algorithm this eigenvector constitutes the initialisation point for the k^{th} hyperplane normal $\mathbf{w}_k \in \mathbb{R}^C$. The position of this hyperplane is further refined via gradient descent from the gradient of a sigmoid smoothed relaxation of Equation 2.45. The remaining hashing hyperplanes are then learnt by updating the residue matrix and sequentially repeating the eigenvector initialisation and gradient descent refinement steps for each.

Despite being a non-linear model, KSH maintains a computationally tractable optimisation algorithm with time complexity $O(NCK + N_{trd}^2CK + N_{trd}C^2K + C^3K)$ by limiting the number C, N_{trd}[18] of sampled data-points used to construct the hash functions and by making a continuous (real-valued) approximation to the binary hashcodes. KSH does not enforce constraints E_3-E_4 of an effective hashcode, but does ensure E_1, E_2 with highly discriminative hashcodes and fast out-of-sample-extension to unseen query data-points.

2.6.4.4 Self-Taught Hashing (STH)

Self-taught hashing (STH) (Zhang et al. (2010b)) employs a two-step procedure for learning the hashing hyperplanes. The first step involves a Laplacian Eigenmap dimensionality reduction which is followed by a second step that learns the hyperplanes for out-of-sample extension to unseen query data-points. STH is therefore reminiscent of the unsupervised data-dependent hashing models Anchor Graph Hashing (AGH) (Section 2.6.3.4) and Spectral Hashing (SH) (Section 2.6.3.2). The first step of STH is identical to that of SH in which K graph Laplacian eigenvectors are extracted from the graph Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{S}$. I present the now familiar graph Laplacian optimisation objective in Equation 2.46

$$\begin{aligned} \text{argmin}_{\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}} \quad & tr(\mathbf{Y}^\intercal(\mathbf{D} - \mathbf{S})\mathbf{Y}) \\ \text{subject to } & \mathbf{Y} \in \mathbb{R}^{N_{trd} \times K} \\ & \mathbf{Y}^\intercal \mathbf{D} \mathbf{1} = \mathbf{0} \\ & \mathbf{Y}^\intercal \mathbf{D} \mathbf{Y} = N_{trd} \mathbf{I}^{K \times K} \end{aligned} \tag{2.46}$$

where $tr(A) = \sum_i A_{ii}$ is the trace operator, $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ is a neighbourhood graph formed from class labels, if two data-points share at least one class in common then $S_{ij} = 1$, otherwise $S_{ij} = 0$ and D is the diagonal degree matrix $D_{ii} = \sum_j S_{ij}$. For computational tractability $N_{trd} \ll N$. Note the slight difference in the constraints between Equation 2.46 and the objective of SH (Equation 2.28). The diagonal degree matrix $\mathbf{D}$ makes an appearance in the constraints of Equation 2.46, which gives a normalised cut of $\mathbf{S}$ rather than a ratio-cut (Aggarwal and Reddy (2014)) when the graph Laplacian eigenvectors are binarised. Equation 2.46 is therefore equivalent to the Laplacian Eigenmap embedding (Belkin and Niyogi (2003)). The solutions of Equation 2.46 are the eigenvectors corresponding to the lowest eigenvalues of the generalised eigenvalue

[18] Typically $N_{trd} = 1000$ and $C = 300$.

problem $L\mathbf{y}^k = \lambda \mathbf{D}\mathbf{y}^k$. The eigenvalue problem can be solved in $O(MN_{trd}^2K)$ operations using M iterations of the Lanczos algorithm (Golub and Van Loan (1996), Zhang et al. (2010b)). Note that, as for BRE (Section 2.6.4.2), it is also straightforward to frame STH as an unsupervised hashing model by computing $\mathbf{S}$ using, for example, the Euclidean distance between feature vectors in the input feature space. In the same way to SH, solving Equation 2.46 approximately preserves properties E_3 and E_4 of an effective hashcode (Section 2.6.1).

Equation 2.46 can be solved as a standard eigenvalue problem to extract the required K graph Laplacian eigenvectors $\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}$. As I discussed in the context of AGH, the spectral embedding matrix must be binarised to form the hashcodes, and only then provides the encoding for the N_{trd} data-points that formed the neighbourhood graph $\mathbf{S}$. Rather than appealing to the Nyström method (Bengio et al. (2004); Williams and Seeger (2001)), as in AGH (Section 2.6.3.4) or making a separable uniform distribution approximation as for SH (Section 2.6.3.2), STH makes the novel contribution of learning a set of K binary support vector machine (SVM) classifiers that predict the bits in the binarised spectral embedding matrix with maximum margin. The learnt classifiers provide the required hyperplane normal vectors $\left\{\mathbf{w}_k \in \mathbb{R}^D\right\}_{k=1}^K$ necessary for out-of-sample extension to unseen data-points. Training K linear SVMs takes $O(N_{trd}DK)$ time (Joachims (2006)) while out-of-sample extension (test time) is $O(DK)$ for a single test data-point.

2.6.4.5 A Brief Summary

I have reviewed four of the most prevalent methods in the literature for injecting a supervised signal into the learning of the hashing hyperplanes for unimodal ANN search. The methods reviewed included ITQ+CCA (Section 2.6.4.1), Binary Reconstructive Embedding (BRE) (Section 2.6.4.2), Supervised Hashing with Kernels (KSH) (Section 2.6.4.3) and Self Taught Hashing (STH) (Section 2.6.4.4). The underlying principle behind all of these methods is to learn a set of K hyperplanes that are informed by must-link or cannot-link constraints on data-point pairs. The hyperplanes should not partition must-link pairs, but should partition cannot-link pairs into distinct hashtable buckets. An example must-link constraint would be for two images of a cat to be placed in the same bucket, while a cannot-link constraint would demand that an image of a dog be placed in a separate bucket. We saw how these methods differ at a high-level only in how the available labels are compared to the projections/hashcodes so as to compute an error signal for further adjustment of the hashing hyperplanes. KSH and

BRE, for example, seek to minimise the difference between the label and either the inner product of the projections of the two data-points (KSH) or the Hamming distance between their binarised hashcodes (BRE). Despite the conceptual similarity between the objective functions, the optimisation algorithms used in their solution were substantially different and formed perhaps the most interesting point of departure between the different hashing models reviewed in this section. BRE for example attempted to optimise the hashing hyperplanes by remaining within the discrete hashcode space, thereby directly tackling an NP-hard optimisation problem. In contrast, KSH relaxed the objective into a continuous domain and used a gradient descent procedure to learn the hashing hyperplanes.

2.6.5 Cross-Modality Projection Methods

Locality sensitive hashing (LSH) and its kernelised variant SKLSH which were both described in Section 2.4 and Section 2.6.2.1 and the data-dependent hashing models presented in Sections 2.6.3-2.6.4 are all confined to *unimodal* retrieval where the queries and the database have identical feature representations. This means that the learnt hyperplanes only partition (bucket) the data-space from that single feature representation. This is a rather limiting restriction of many existing hashing models because much of the data found today, particularly on the internet, is associated with multiple modalities[19]. For example, consider an image from the popular photo sharing website Flickr[20] which is not only described by the raw pixel values themselves, but also with associated tags assigned by users and geolocation information sourced from the GPS system on the camera. It would clearly be very useful if we could pose a query in the form of an image and retrieve relevant tags (Figure 2.19), or give the retrieval system geographical coordinates and receive images related to that locality.

The data-dependent hashing models I describe in this section are able to hash related data-points existing across two modalities into the same hashtable buckets, thereby bringing the computational advantages of approximate nearest neighbour search to multi-modal retrieval. Denote as $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D_x}$ the feature descriptors in modality $\mathcal{X}$ and $\mathbf{Z} \in \mathbb{R}^{N_{trd} \times D_z}$ the feature descriptors in modality $\mathcal{Z}$, where usually the dimensionalities are not equal $D_x \neq D_z$. For simplicity of description I assume that both datasets have the same number of training data-points N_{trd}, and I further denote as N_{xz} the number of *paired* data-points across the modalities ($N_{xz} \leq N_{trd}$). The logical relationship

[19] I use the term ‘modality’ and ‘feature space’ interchangeably in this thesis.
[20] `http://www.flickr.com`

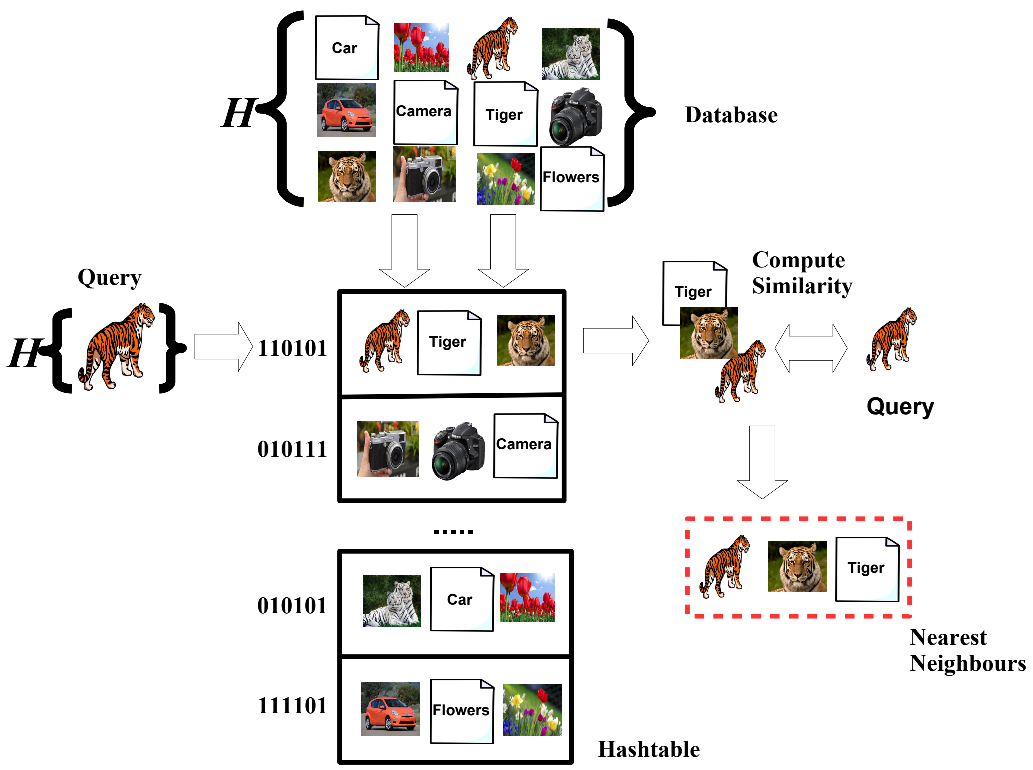
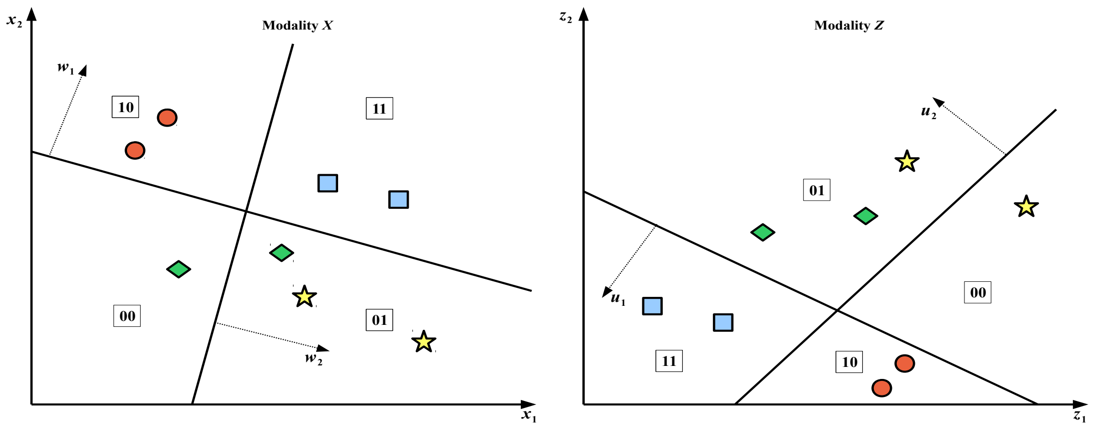


Figure 2.19: Cross-modal hashing-based ANN search. In the cross-modal variant of hashing-based ANN search we wish to partition the input-space such that similar data-points across modalities fall into the same hashtable buckets. In this diagram I show how cross-modal hash functions can be used to retrieve similar images and documents to a query image in constant time. The cross-modal hash functions $\mathcal{H}$ assign similar hashcodes to similar images and documents thereby allowing similar data-points in different modalities to collide in the same hashtable buckets.

between the data-points is encoded in an adjacency matrix $\mathbf{S} \in \{0,1\}^{N_{xz} \times N_{xz}}$, where $S_{ij} = 1$ indicates that pair $(\mathbf{x}_i, \mathbf{z}_j)$ are related, and 0 otherwise. At a high level all five of these models attempt to learn *two* sets of K hyperplanes denoted as $\mathbf{W} \in \mathbb{R}^{D_x \times K}$ and $\mathbf{U} \in \mathbb{R}^{D_z \times K}$, one set for feature space $\mathcal{X}$ and another for feature space $\mathcal{Z}$, such that similar data-points ($S_{ij} = 1$) *across* the two modalities receive similar hashcodes $d_{hamm}(g_{\mathcal{X}}(\mathbf{x}_i), g_{\mathcal{Z}}(\mathbf{z}_j)) \approx 0$, and vice-versa for dissimilar data-points ($S_{ij} = 0$). Here $\left\{g_x : \mathbb{R}^D \rightarrow \{0,1\}^K\right\}$ is the binary embedding function formed from the concatenation of K hash functions $\{h_k^{\mathcal{X}} : \mathbb{R}^D \rightarrow \{0,1\}\}_{k=1}^{K}$ for modality $\mathcal{X}$, and similar for modality $\mathcal{Z}$. This is a logical extension of the unimodal case in which we not only wish to make similar data-points *within* a modality fall into the same hashtable buckets (e.g. two images of a cat), but also similar data-points *across* the two modalities (e.g. an image of



(a) **Modality $\mathcal{X}$** (b) **Modality $\mathcal{Y}$**

Figure 2.20: The essence of learning to hash across modalities. In Figure (a) I show the first modality $\mathcal{X}$ (e.g. image descriptor space) with data-points $\left\{\mathbf{x}_i \in \mathbb{R}^{D_x}\right\}_{i=1}^{N}$ and hyperplane normal vectors $\left\{\mathbf{w}_k \in \mathbb{R}^{D_x}\right\}_{k=1}^{K}$. In Figure (b) I show a different feature space $\mathcal{Z}$ (e.g. textual annotations) with data-points $\left\{\mathbf{z}_i \in \mathbb{R}^{D_z}\right\}_{i=1}^{N}$ and hyperplane normal vectors $\left\{\mathbf{u}_k \in \mathbb{R}^{D_z}\right\}_{k=1}^{K}$. Similar data-points within and across modalities are indicated by the same colour and shape. The goal of cross-modal hashing is to position the two sets of hyperplanes in such a way that they assign the same hashcodes to the same data-points both within and across the two modalities.

a cat and a text snippet describing a cat). In Section 2.4 and Sections 2.6.3-2.6.4, I discussed how to learn K hyperplanes that assign similar data-points similar hashcodes in the *same* modality. I saw how this is achieved by positioning the hashing hyperplanes in the input space in a way that attempts to maximise the number of true nearest neighbours within the same buckets. In this section we will see how this notion can be extended to learning *two sets* of K hyperplanes that generate similar hashcodes for related data-points in two different modalities. In practice this boils down to augmenting the objective function with a *consistency* term that ensures the two sets of hyperplanes agree on their hashcode output for similar cross-modal data-points. I provide an intuitive high-level overview of this fundamental concept in Figure 2.20.

I use the same strategy, namely a freely available codebase and widely referenced publication, to select appropriate baselines for cross-modal retrieval as I did for the unimodal baselines described in Sections 2.6.3-2.6.4. The selected baselines are the seminal Cross View Hashing (CVH) model of Kumar and Udupa (2011) (Section 2.6.5.1), Co-Regularised Hashing (CRH) (Zhen and Yeung (2012)) (Section 2.6.5.2), Predicable Dual View Hashing (PDH) (Rastegari et al. (2013)) (Section 2.6.5.4), Inter-

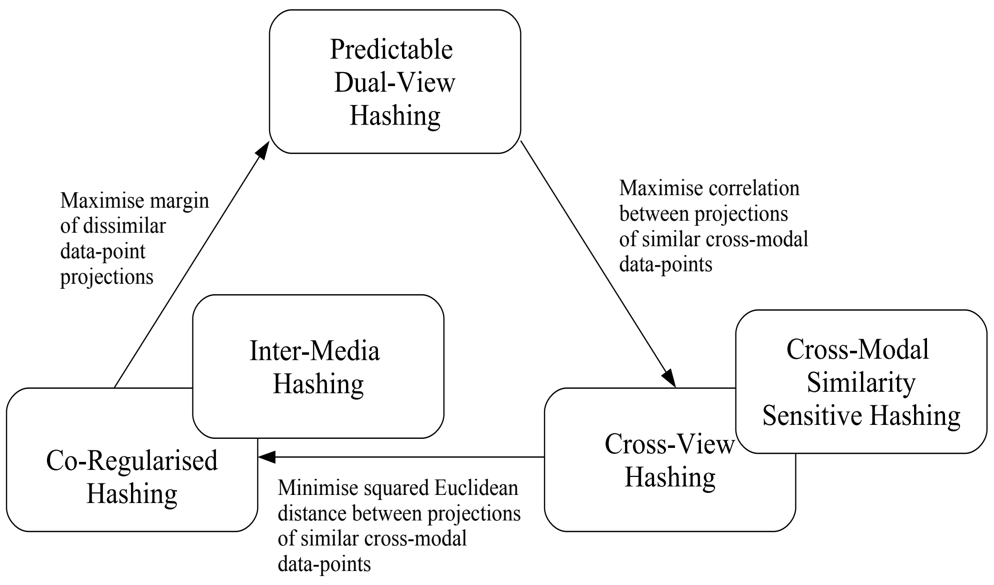

Predictable Dual-View Hashing
Maximise margin of dissimilar data-point projections
Maximise correlation between projections of similar cross-modal data-points
Inter-Media Hashing
Cross-Modal Similarity Sensitive Hashing
Co-Regularised Hashing
Cross-View Hashing
Minimise squared Euclidean distance between projections of similar cross-modal data-points

Figure 2.21: Relationship between the five cross-modal hashing models reviewed in this section. I only consider the inter-modal consistency term when relating the models (ignoring intra-modal and out-of-sample extension terms). The labels on the arcs denote the essential transform required to convert one model into the model(s) pointed to by the arc.

Media Hashing (IMH) (Song et al. (2013)) (Section 2.6.5.5) and Cross Modal Semi-Supervised Hashing (CMSSH) (Bronstein et al. (2010)) (2.6.5.3).

2.6.5.1 Cross View Hashing (CVH)

Cross View Hashing (CVH)[21] (Kumar and Udupa (2011)) is equivalent to $ITQ+CCA$ (Section 2.6.4.1) in its use of Canonical Correlation Analysis (CCA) to find two sets of hyperplanes that maximise the correlations of the projections from two different modalities. There are two differences to $ITQ+CCA$: firstly, CVH retains both sets of hyperplane normals $\mathbf{W} \in \mathbb{R}^{D_x \times K}$ and $\mathbf{U} \in \mathbb{R}^{D_z \times K}$, rather than only using the set pertaining to the visual modality; secondly, CVH does not involve a post-processing step that rotates the input feature space to balance the variance captured across the hyperplanes. The hash function for CVH is the standard linear hash function. Equation 2.47 presents the hash functions for both modalities

[21] As is standard in the literature I consider the special case of CVH where only cross-modality supervision is available and each data-point is paired with only one other in the opposing modality (Section 3.2 in Kumar and Udupa (2011)).

$$\begin{aligned} h_k^{\mathcal{X}}(\mathbf{x}_i) &= \frac{1}{2}(1+sgn(\mathbf{w}_k^{\intercal}\mathbf{x}_i)) \\ h_k^{\mathcal{Z}}(\mathbf{z}_i) &= \frac{1}{2}(1+sgn(\mathbf{u}_k^{\intercal}\mathbf{z}_i)) \end{aligned} \tag{2.47}$$

Following the same argument as for ITQ+CCA, the asymptotic computational complexity of CVH is $O(N_{trd}D^2 + D^3)$ where $D = max(D_x, D_z)$. CVH was one of the first proposed cross-modal hashing models to be proposed in the literature and typically features in previous research as the de-facto baseline for comparison. The cross-modal hashing models I will review in Sections 2.6.5.2-2.6.5.5 introduce new schemes for learning both sets of hyperplane that achieve a higher retrieval effectiveness than CVH on standard image-text datasets.

2.6.5.2 Co-Regularised Hashing (CRH)

Co-Regularised Hashing (CRH) learns $2K$ cross-modal hash functions by solving K individual max-margin optimisation problems sequentially (Zhen and Yeung (2012)). Boosting (Freund and Schapire (1997)) is used in each step to coordinate the learning of the hash functions so that the pairwise constraints not met by hyperplanes constructed earlier in the optimisation sequence have a gradually higher likelihood of being met by subsequent hyperplanes. This brings about a dependence between the bits, a trait we have seen before in the context of the unimodal data-dependent hashing model KSH (Section 2.6.4.3). CRH uses the standard linear hash function (Equation 2.47) as for CVH (Section 2.6.5.1). The objective function for learning the two hyperplane normals $\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}$ pertaining to the same bit in modalities $\mathcal{X}$, $\mathcal{Z}$ is made up of three main terms: one to position the hyperplane normal $\mathbf{w}_k$ in modality $\mathcal{X}$, another to position the hyperplane normal $\mathbf{u}_k$ in modality $\mathcal{Z}$ and a third consistency term that forces both hyperplanes to give similar projections for related data-points. The CRH objective is presented in Equation 2.48

$$\begin{aligned} \text{argmin}_{\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}} \quad & \frac{1}{N_x}\sum_{i=1}^{N_x}[1 - |\mathbf{w}_k^{\intercal}\mathbf{x}_i|]_+ + \frac{1}{N_z}\sum_{j=1}^{N_z}[1 - |\mathbf{u}_k^{\intercal}\mathbf{z}_j|]_+ \\ & \gamma \sum_{i,j=1}^{N_{xz}} \alpha_{ij} S_{ij}(\mathbf{w}_k^T\mathbf{x}_i - \mathbf{u}_k^{\intercal}\mathbf{z}_j)^2 + \frac{\lambda_x}{2}\|\mathbf{w}_k\|^2 + \frac{\lambda_z}{2}\|\mathbf{u}_k\|^2 \end{aligned} \tag{2.48}$$

where $\left\{\alpha_{ij} \in \mathbb{R}_+\right\}_{i,j=1}^{N_{trd}}$ are weights updated using Adaboost (Freund and Schapire (1997)), $\lambda_x \in \mathbb{R}_+, \lambda_z \in \mathbb{R}_+$ are regularisation constants, $[a]_+$ is equal to a if $a \geq 0$, and 0 otherwise, and $\gamma \in \mathbb{R}_+$ is an scalar governing the importance of the cross-modal term. The intra-modality loss terms guide the projections to be away from zero by a margin so that the data-points do not lie too close to the dividing hyperplanes, thereby encouraging generalisability of the hash functions. The inter-modal loss term is intuitive in its attempt to minimise the squared difference between the projections of similar data-points across modalities[22].

Equation 2.48 is non-convex and so Zhen and Yeung (2012) minimise it in an alternate manner by solving two sub-problems: fixing $\mathbf{w}_k \in \mathbb{R}^{D_x}$ and optimising for $\mathbf{u}_k \in \mathbb{R}^{D_z}$ and vice-versa. In practice this is achieved using the Concave-Convex Procedure (CCVP) (Yuille and Rangarajan (2003)) by framing each sub-problem as a difference of convex functions. Having learnt hyperplane normals $\mathbf{w}_k$, $\mathbf{u}_k$ at the k^{th} step, the weights $\{\alpha_i \in \mathbb{R}_+\}_{i=1}^{N_{trd}}$ for the point-pairs are updated using the standard Adaboost framework (Freund and Schapire (1997)), in which the error term for Adaboost is based upon a count of the number of times the outputs of the cross-modal hash functions disagree. This entire procedure is then repeated with Equation 2.48 solved for hyperplanes $\mathbf{w}_{k+1}$, $\mathbf{u}_{k+1}$ using the updated Adaboost weights. The computational time complexity of CRH is bounded by $O(KMND)$ where $D = max(D_x, D_z)$ and M is the number of iterations required for convergence of the Pegasos solver (Shalev-Shwartz et al. (2007)). Properties E_1, E_2 of an effective hashcode (Section 2.6.1) are preserved by CRH but not properties E_3, E_4.

2.6.5.3 Cross-Modal Similarity Sensitive Hashing (CMSSH)

Cross-Modal Similarity Sensitive Hashing (CMSSH) (Bronstein et al. (2010)) presents a considerably simpler optimisation framework compared to CRH (Section 2.6.5.2) focusing entirely on ensuring the output of the cross-modal hash functions are consistent with each other, but without any specific terms for optimising the intra-modal similarity. CMSSH learns the $2K$ hash functions using a sequential procedure where, at the k^{th} step, two hyperplane normal vectors $\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}$ are computed with the weighted objective function presented in Equation 2.49 that effectively coerces the k^{th} pair of hash functions to correct mistakes committed by the k-1 previous hash functions

$$\text{argmax}_{\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}} \sum_{i,j=1}^{N_{xz}} \alpha_{ij} S_{ij} sgn(\mathbf{w}_k^\intercal \mathbf{x}_i) sgn(\mathbf{u}_k^\intercal \mathbf{z}_j) \tag{2.49}$$

[22] In practice CRH also has an additional term that pushes dissimilar points further apart. I omit this here for clarity and conciseness of explanation.

where $\{\alpha_{ij} \in \mathbb{R}_+\}_{i,j=1}^{N_{xz}}$ are per data-point pair weights adjusted using Adaboost (Freund and Schapire (1997)). CMSSH is similar to CVH in that the correlation of the projections of related cross-modal data-points is maximised. As it stands Equation 2.49 is discontinuous and therefore non-differentiable making it difficult to optimise with the sign function intact. Bronstein et al. (2010) therefore make the standard spectral relaxation which we have previously seen in many other data-dependent hashing models such as KSH (Section 2.6.4.3) and SH (Section 2.6.3.2). This relaxation simply involves dropping the sign function altogether giving Equation 2.50.

$$\begin{aligned} \operatorname{argmax}_{\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}} & \sum_{i,j=1}^{N_{xz}} \alpha_{ij} S_{ij} (\mathbf{w}_k^\intercal \mathbf{x}_i)(\mathbf{u}_k^\intercal \mathbf{z}_j) \\ = & \; \mathbf{w}_k^\intercal \left\{ \sum_{ij=1}^{N_{xz}} \alpha_{ij} S_{ij} \mathbf{x}_i \mathbf{z}_j^\intercal \right\} \mathbf{u}_k \\ = & \; \mathbf{w}_k^\intercal \mathbf{C} \mathbf{u}_k \end{aligned} \tag{2.50}$$

Equation 2.50 can be solved in closed form by performing an SVD on the matrix $\mathbf{C} \in \mathbb{R}^{D_x \times D_z}$ taking $O(D_x^2 D_z + D_x D_z^2 + D_z^3)$ operations[23]. The per pair weights $\{\alpha_{ij} \in \mathbb{R}_+\}_{i,j=1}^{N_{xz}}$ are then updated using Adaboost to emphasise the misclassified data-point pairs and the step repeated for the k+1 set of hyperplanes. The learnt hash functions are used in the standard linear hash function (Equation 2.47) to generate binary hashcodes for multimodal data. In a similar manner to CRH, CMSSH maintains properties E_1, E_2 of an effective hashcode, but does not seek to conserve property E_4 due to the sequential dependence of hashcode bits induced through boosting.

2.6.5.4 Predictable Dual-View Hashing (PDH)

Predictable Dual-View Hashing (PDH) (Rastegari et al. (2013)) is the closest cross-modal hashing model to my own contribution outlined in Chapter 6 and in Moran and Lavrenko (2015b). Given this close relationship I present the full specification of the PDH model in Algorithm 5 so that it is straightforward to compare and contrast both models. Similar to CRH and CMSSH, PDH solves for the $2K$ hashing hyperplanes sequentially by solving for a pair of hyperplane normal vectors $\mathbf{w}_k \in \mathbb{R}^{D_x}, \mathbf{u}_k \in \mathbb{R}^{D_z}$ at the k^{th} step. PDH solves for the hyperplanes using an SVM-based formulation in which

[23]As $\mathbf{C} \in \mathbb{R}^{D_x \times D_z}$ may be non-square the solution is obtained via an SVD rather than the standard eigenvalue problem used for the unimodal data-dependent hashing models described in Section 2.6.3.

the hyperplanes are trained to partition data-points with opposing bits with maximum margin. Different to these previously reviewed models, PDH does not induce a dependence between bits using boosting but instead explicitly attempts to enforce property E_4 of an effective hashcode, namely that the bits should be pairwise independent. I present the PDH objective function in Equation 2.51

$$\begin{aligned}
\text{argmin}_{\mathbf{W}\in\mathbb{R}^{D_x\times K},\mathbf{U}\in\mathbb{R}^{D_z\times K}} \quad & ||\mathbf{B}^x\mathbf{B}^x - \mathbf{I}||_2^2 + ||\mathbf{B}^z\mathbf{B}^z - \mathbf{I}||_2^2 + \sum_{k=1}^{K} ||\mathbf{w}_k||^2 + \sum_{k=1}^{K} ||\mathbf{u}_k||^2 \\
& + C_x \sum_{i,k=1}^{N_{trd}} \xi_{ik}^x + C_z \sum_{i,k=1}^{N_{trd}} \xi_{ik}^z \\
\text{subject to } & \mathbf{B}^x = sgn(\mathbf{XW}) \\
& \mathbf{B}^z = sgn(\mathbf{ZU}) \\
& b_{ik}^z(\mathbf{w}_k^\intercal \mathbf{x}_i) \geq 1 - \xi_{ik}^x \\
& b_{ik}^x(\mathbf{u}_k^\intercal \mathbf{z}_i) \geq 1 - \xi_{ik}^z
\end{aligned} \tag{2.51}$$

where $\xi_{ik}^x \in \mathbb{R}$, $\xi_{ik}^z \in \mathbb{R}$ are slack variables that allow some points $\mathbf{x}_i, \mathbf{z}_i$ to fall on the wrong side of hyperplanes with normal vectors $\mathbf{w}_k, \mathbf{u}_k$ and $C_x \in \mathbb{R}_+$, $C_z \in \mathbb{R}_+$ are parameters that permit a trade off between the size of the margins $\frac{1}{||\mathbf{w}_k||}, \frac{1}{||\mathbf{u}_k||}$ against the number of points misclassified by $\mathbf{w}_k, \mathbf{u}_k$. The first two terms of the objective function enforce the constraint that the bits should be pairwise independent (property E_3). The last two constraints are reminiscent of the standard SVM max-margin objective with the hashcode bits b_{ik}^x, b_{ik}^z in this case acting as the requisite target labels. Note the subtle but important feature of these constraints where the hashcode bits (b_{ik}^x, b_{ik}^z) for one feature space are used as the targets for hyperplanes existing in the other feature space. This means that over multiple iterations the hyperplanes in both feature spaces should become more consistent in their projections for similar and dissimilar data-points.

Rastegari et al. (2013) solve Equation 2.51 by dividing it into multiple steps as highlighted in Algorithm 5. Firstly, using the bits of the hashcodes in $\mathbf{B}^x \in \{-1,1\}^{N_{trd}\times K}$, $\mathbf{B}^z \in \{-1,1\}^{N_{trd}\times K}$ are used as the labels to train $2K$ SVM classifiers (Lines 6, 10 in Algorithm 5). This step computes an initial estimate of hashing hyperplanes $\mathbf{W} \in \mathbb{R}^{D_x\times K}$, $\mathbf{U} \in \mathbb{R}^{D_z\times K}$. The bits in $\mathbf{B}^x, \mathbf{B}^z$ are then re-labelled with the learnt SVMs (Lines 9, 13 in Algorithm 5), which flips the sign of those data-points that happened to fall on the wrong side of the respective hyperplanes. The pairwise independence property between the bits is approximately enforced by solving the familiar graph Laplacian eigenvalue problem in Equation 2.52

Algorithm 5: PREDICTABLE DUAL VIEW HASHING (PDH) (RASTEGARI ET AL. (2013))

```
Input: Data-points $\mathbf{X} \in \mathbb{R}^{N_{trd} \times D_x}$, $\mathbf{Z} \in \mathbb{R}^{N_{trd} \times D_z}$, Iterations $M$
Output: Hyperplanes $\mathbf{W} \in \mathbb{R}^{D_x \times K}$, $\mathbf{U} \in \mathbb{R}^{D_z \times K}$
1  Initialise $\mathbf{W}$, $\mathbf{U}$ via CCA from $\mathbf{X}$, $\mathbf{Z}$
2  for $m \leftarrow 1$ to $M$ do
3      for $k \leftarrow 1$ to $K$ do
4          $\mathbf{b}^x_k = \mathbf{B}^x_{\bullet k}$
5          $\mathbf{b}^z_k = \mathbf{B}^z_{\bullet k}$
6          Train SVM$^x_k$ with $\mathbf{b}^z_k$ as labels, training dataset $\mathbf{X}$
7          Obtain hyperplane $\mathbf{h}^x_k$
8          $\mathbf{W}_{\bullet k} = \mathbf{w}_k$
9          $\mathbf{B}^x_{\bullet k} = sgn(\mathbf{X}\mathbf{w}_k)$
10         Train SVM$^z_k$ with $\mathbf{b}^x_k$ as labels, training dataset $\mathbf{Z}$
11         Obtain hyperplane $\mathbf{h}^z_k$
12         $\mathbf{U}_{\bullet k} = \mathbf{u}_k$
13         $\mathbf{B}^z_{\bullet k} = sgn(\mathbf{Z}\mathbf{u}_k)$
14     end
15     Update $\mathbf{B}^x$, $\mathbf{B}^z$ by solving eigenvalue problem in Equation 2.52.
16 end
17 return $\mathbf{W}, \mathbf{U}$
```

$$
\begin{aligned}
\operatorname{argmin}_{\mathbf{Y} \in \mathbb{R}^{N_{trd} \times K}} \quad & tr(\mathbf{Y}_l^\intercal(\mathbf{D}_l - \mathbf{S}_l)\mathbf{Y}_l) \\
\text{subject to } & \mathbf{Y}_l \in \mathbb{R}^{N_{trd} \times K} \\
& \mathbf{Y}_l^\intercal \mathbf{1} = 0 \\
& \mathbf{Y}_l^\intercal \mathbf{Y}_l = N_{trd} \mathbf{I}^{K \times K}
\end{aligned}
\tag{2.52}
$$

where $\mathbf{Y}_l \in \mathbb{R}^{N_{trd} \times K}$ for $l \in \{x, z\}$ are the real-valued (unbinarised) projections for modalities $\mathcal{X}, \mathcal{Z}$ and $\mathbf{S}_l = \mathbf{Y}_l^\intercal \mathbf{Y}_l$, with $\mathbf{D}_{ii} = \sum_j S_{ij}$. As we saw in Section 2.6.3, the solution to this problem is the top K eigenvectors with minimal eigenvalues (Line 15). These K eigenvectors are subsequently binarised to form the updated hashcodes. Intuitively this eigenvalue problem is attempting to lower the pairwise correlation between the data-point projection vectors along the rows of $\mathbf{Y}_l$ for $l \in \{x, z\}$ while maintaining the relative distances between the projection vectors as defined by the inner product similarity. These steps are repeated M times until the algorithm has reached a suitable convergence point, at which point the learnt hyperplanes can be used in the standard linear hash function (Equation 2.47) to hash novel cross-modal data-points into hashtable buckets. The training time complexity is dominated by the $O(MN_{trd}^2K)$ operations to solve the eigenvalue problems across M iterations using the Lanczos algorithm (Golub and Van Loan (1996)).

2.6.5.5 Inter-Media Hashing (IMH)

Inter-Media Hashing (IMH) (Song et al. (2013)) can be thought of as a semi-supervised cross-modal hashing model which not only utilises the pairwise supervisory information in the adjacency matrix $\mathbf{S}^{xz} \in \{0,1\}^{N_{xz} \times N_{xz}}$, but also from unsupervised information originating from all N_{trd} data-points *within* each modality, that is, including those data-points that do not feature in $\mathbf{S}^{xz}$. The relationship between these data-points is computed through construction of a Euclidean k-NN graph within each modality. Denote as $\mathbf{S}^{x} \in \{0,1\}^{N_{trd} \times N_{trd}}$ the k-NN graph between data-points in modality $\mathcal{X}$, and similarly for modality $\mathcal{Z}$ where $\mathbf{S}^{z} \in \{0,1\}^{N_{trd} \times N_{trd}}$. Using modality $\mathcal{X}$ as an example, the k-NN graph is constructed as in Equation 2.53

$$S_{ij}^{x} = \begin{cases} 1 & \text{if } \mathbf{x}_i \in NN_k(\mathbf{x}_j) \text{ or } \mathbf{x}_j \in NN_k(\mathbf{x}_i) \\ 0 & otherwise \end{cases} \tag{2.53}$$

where $NN_k(\mathbf{x}_i)$ is a function that returns the set of k-nearest neighbours for data-point $\mathbf{x}_i$ as measured under, for example the Euclidean distance metric.

The IMH semi-supervised approach is to be contrasted with CVH, CMSSH and PDH which learn entirely from the supervisory information in the adjacency matrix $\mathbf{S}$ that pairs related data-points across the two modalities. In this sense IMH, in its full form, bears most resemblance to CRH (Section 2.6.5.2) which also proposes unsupervised terms in the objective function that act as a form of regularisation during the training procedure. When the learning is entirely confined to the labelled data-points in $\mathbf{S}^{xz}$ and further $\mathbf{S}^{xz} = \mathbf{I}^{N_{xy} \times N_{xy}}$, Song et al. (2013) show that IMH is in fact equivalent to the CCA-based CVH model (Section 2.6.5.1). The IMH objective function is presented in Equation 2.54

$$
\begin{aligned}
\operatorname{argmin}_{\mathbf{W},\mathbf{U},\mathbf{Y}^x,\mathbf{Y}^z} \quad & \lambda \sum_{i,j=1}^{N_{trd}} S^x_{ij} \|\mathbf{y}^x_i - \mathbf{y}^x_j\|_2^2 + \lambda \sum_{i,j=1}^{N_{trd}} S^z_{ij} \|\mathbf{y}^z_i - \mathbf{y}^z_j\|_2^2 + \sum_{i,j=1}^{N_{xz}} S^{xz}_{ij} \|\mathbf{y}^x_i - \mathbf{y}^z_j\|_2^2 \\
& + \sum_{i=1}^{N_{trd}} \|\mathbf{W}^\intercal \mathbf{x}_i - \mathbf{y}^x_i\|_2^2 + \beta \|\mathbf{W}\|_F^2 + \sum_{j=1}^{N_{trd}} \|\mathbf{U}^\intercal \mathbf{z}_j - \mathbf{y}^z_j\|_2^2 + \beta \|\mathbf{U}\|_F^2 \\
& \text{subject to } (\mathbf{Y}^x)^\intercal \mathbf{Y}^x = \mathbf{I}^{D_x \times D_x} \\
& (\mathbf{Y}^x)^\intercal \mathbf{1} = \mathbf{0}
\end{aligned}
\tag{2.54}
$$

where $\mathbf{S}^{xz} = \mathbf{I}^{N_{xz} \times N_{xz}}$ and $\beta \in \mathbb{R}, \lambda \in \mathbb{R}$ are user-specified scalar parameters affecting the importance of the different terms in the objective function.

The IMH objective function is quite intuitive and builds on previous research such as Weiss et al. (2008); Kumar and Udupa (2011); Zhen and Yeung (2012). The first two terms encourage similar data-points within both modalities to have similar projected values and therefore similar hashcodes upon binarisation. It is exactly the graph Laplacian eigenvalue problem (Equation 2.28) we first saw in the context of Spectral Hashing (SH) in Section 2.6.3.2 and extended first to the dual-modality case by CVH (Section 2.6.5.1). The third term encourages the projections of similar data-points across modalities to be the same, which is akin to the inter-modal consistency term used in the CRH model (Section 2.6.5.2) without the Adaboost per-pair weights. The last two terms are out-of-sample extension terms yielding the desired hashing hyperplanes $\mathbf{W} \in \mathbb{R}^{D_x \times K}, \mathbf{U} \in \mathbb{R}^{D_z \times K}$ using the standard L_2 regularised linear regression formulation with the learnt projections for the N_{trd} training data-points as the regression targets. Song et al. (2013) minimise Equation 2.54 by transforming the objective into a trace minimisation problem involving the modality $\mathcal{X}$ projections $\mathbf{Y}^x$. As is customary in the learning to hash literature (Section 2.6.3), the trace minimisation is solved as an eigenvalue problem taking the K eigenvectors with the smallest eigenvalues as the columns of $\mathbf{Y}^x$. Note that solving this eigenvalue problem will achieve the orthogonality constraint in the objective function. Given the solution for the projections in modality $\mathcal{X}$, Song et al. (2013) show that the optimal $\mathbf{Y}^z$ and $\mathbf{W}$, $\mathbf{U}$ can be obtained using closed form formulae based upon the learnt $\mathbf{Y}^x$.

As for all hashing models relying on a matrix factorisation, solving the eigenvalue problem dominates the computational time complexity of IMH requiring $O(N_{trd}^3)$ operations[24]. IMH maintains properties E_1, E_2 of an effective hashcode in both modalities

[24]For the datasets we consider in this thesis, $N_{trd} = 2{,}000$-$10{,}000$ to constrain computation time. In a real-world application N_{trd} be significantly higher.

$\mathcal{X}$, $\mathcal{Y}$, while only maintaining properties E_3, E_4 in modality $\mathcal{X}$ as a result of solving the eigenvalue problem.

2.6.5.6 A Brief Summary

In this section I described five prominent hashing algorithms that are capable of indexing similar data-points that exist in two incommensurable feature spaces into the same hashtable buckets. Specifically, I reviewed Cross-View Hashing (CVH) (Section 2.6.5.1), Co-Regularised Hashing (Section 2.6.5.2), Cross-Modal Semi-Supervised Hashing (Section 2.6.5.3), Predictable Dual-View Hashing (Section 2.6.5.4) and Inter-Media Hashing (Section 2.6.5.5). The essential link between all of these algorithms was the learning of two sets of K hyperplanes, one set of K hyperplanes for each feature space, in a way that encourages the hyperplanes in both spaces to assign similar projected values to similar cross-modal data-points. In all cases this is achieved by framing an optimising an objective function with a cross-modal consistency term that penalises a mismatch between the projected values of similar cross-modal data-points. Some of the more recently proposed cross-modal hashing algorithms (CRH, IMH, PDH) augmented this inter-modal consistency term with additional intra-modal terms that regularise the learning of the hyperplanes by, for example, ensuring that similar within-modality data-points receive similar projected values. The disadvantage of all of these algorithms are their reliance on either an expensive matrix factorisation or non-convex optimisation.

2.7 Conclusion

The purpose of this chapter was to introduce the background information relevant to the topic of this thesis, all of which is important for understanding the novel contributions that will be introduced in later chapters. I began this chapter in Section 2.3 by motivating the need for more efficient algorithms for nearest neighbour (NN) search that do not require an exhaustive brute-force scan of the dataset. This led us to the field of approximate nearest neighbour search which I argued is dominated by the seminal method of Locality Sensitive Hashing (LSH). We saw in Section 2.4 how LSH is in fact a family of different algorithms for generating similarity preserving hashcodes for a wide range of similarity functions of interest, from the inner product similarity to the Euclidean distance. I discussed how the LSH hash function family for the inner product similarity forms the focus of this thesis. In this case LSH will generate hashcodes with a low Hamming distance to each other for those data-points that are similar under the inner product similarity. This property enables the hashcodes to be used as indices into the buckets of a set of hashtables to retrieve nearest neighbours in a constant time per query, a much improved query-time versus a brute-force linear scan.

In Sections 2.5-2.6, I then discussed how the contributions in this thesis address the effectiveness of two critical components of the LSH algorithm: projection (hyperplane) learning and binary quantisation. Retrieval effectiveness is highly dependent on how well these two steps preserve the original neighbourhood structure between the data-points in the hashcode Hamming space. Unfortunately, we saw how LSH generates its hyperplanes and quantisation thresholds randomly in the input space relying on asymptotic guarantees that as the number of hyperplanes increases, the desired similarity will be well reflected by the Hamming distance between the binary hashcodes. A random partitioning may lead to the separation of many related data-points into different hashtable buckets, contrary to the central premise of hashing-based ANN search. I described how relaxing this data-independence assumption could mitigate this effect and potentially lead to improved retrieval effectiveness while simultaneously generating more compact hashcodes compared to LSH. This dissertation is not the only research to address this important downside to LSH and so I conducted a review of recently proposed and closely related work within the quantisation and projection branches of the field in Section 2.5 and Section 2.6, respectively. My specific focus was on data-dependent hashing algorithms that learn the hashing hyperplanes and quantisation thresholds in a way that is informed by the distribution of the data, using either an unsupervised (Section 2.6.3) or supervised (Section 2.6.4) signal to avoid placing related unimodal (Sections 2.6.3-2.6.4) or cross-modal (Section 2.6.5) data-points into different hashtable buckets. The reviewed algorithms will form a broad and strong set of baselines in our experimental evaluation presented in later chapters.

Four questions unaddressed by the current literature have been identified by conducting this review, each of which are investigated in subsequent chapters of this thesis. Firstly, the multi-threshold quantisation models that I described in Section 2.5 employ unsupervised learning to position the thresholds. Given this, I explore in Chapter 4 how retrieval effectiveness can be improved by formulating a suitable *semi-supervised* objective function for multiple threshold learning. Secondly, I discovered how most existing quantisation models for hashing are restricted to a *uniform* number of thresholds per projected dimension. To the best of my knowledge there is no existing work that explores the effect of varying the number of assigned thresholds per projected dimension on retrieval effectiveness. I fill this gap in the current literature in Chapter 5 by proposing a new quantisation model that learns an appropriate allocation of thresholds across projected dimensions based on the neighbourhood preserving quality of the associated hyperplanes. Thirdly, the current breed of supervised projection functions identified in Sections 2.6.4-2.6.5 employ computationally expensive eigen-decomposition and kernel-based optimisation strategies. In Chapter 6, I explore the extent to which a simpler and computationally less expensive optimisation algorithm can compete with these state-of-the-art supervised projection functions, both within the unimodal and cross-modal problem domains. Finally, there is no previous work that learns both the hashing hypersurfaces and *multiple* quantisation thresholds in the same model. I address this knowledge gap in Chapter 7 by combining my quantisation models from Chapters 4-5 with my supervised projection function from Chapter 6.

Having now firmly placed the research described in this thesis in the context of previous related work I will introduce in the next chapter the datasets, evaluation paradigms and evaluation metrics that will be used to judge the quality of nearest neighbour search in my experimental evaluation.

Chapter 3

Experimental Methodology

3.1 Introduction

In this chapter I describe the experimental methodology adopted throughout the thesis. This includes the datasets selected as the testbed for the retrieval experiments (Section 3.2), the definition of groundtruth for evaluation (Section 3.3) and the metrics adopted to ascertain retrieval effectiveness (Section 3.6). I attempt to keep the evaluation strategy aligned as closely as possible to previously related work in the learning to hash literature. However, as I will discuss throughout this section, the evaluation methodology used by previous related work is not consistent across publications and exhibits certain flaws in the experimental design. This chapter describes my remedy for these flaws and places the evaluation on a standard foundation.

3.2 Datasets

The focus of this thesis is learning hash functions for the task of large-scale image retrieval. This task will be split into three sub-tasks: 1) an image is used to retrieve related images from a still image archive, 2) a text query is used to retrieve related images, 3) an image query is used to retrieve relevant annotations for that image. I will therefore conduct both unimodal and cross-modal retrieval experiments in this thesis which will cover a wide range of important use-cases in image retrieval from query-by-example search to image annotation. Unimodal datasets are those where the query and the database are in the same visual modality such as bag-of-visual-word feature descriptors. Cross-modal datasets permit retrieval experiments that straddle two different modalities such as a textual query executed against an image database. The latter task mimics the familiar image search scenario offered by many modern web search engines. In all cases I will constrain the evaluation to baselines that perform these tasks using hashing-based ANN search and do not seek to compare against, for example, fully fledged image annotation models. To align the evaluation closely with the learning to hash literature I select a subset of the most popular datasets from both categories as my experimental testbed (Sections 3.2.1-3.2.2). My desiderata for dataset selection is two-fold: firstly, the datasets must be publicly available to enable replication of experimental results by a third party; and secondly the datasets must be standard in the sense that they have been widely used in related publications. This ensures that the experimental results published in this thesis are reproducible and directly comparable to previously published research.

3.2.1 Unimodal Retrieval Experiments

For the unimodal experiments, I select four popular and freely available image datasets: LabelMe, CIFAR-10, NUS-WIDE and SIFT1M. The datasets are of widely varying size (22,019-1 million images), are represented by an array of different feature descriptors (from GIST, SIFT to bag of visual words) and cover a diverse range of different image topics from natural scenes to personal photos, logos and drawings. These properties ensure that the datasets will provide a challenging test suite for evaluation in this thesis. All datasets are identical to those used in many recent publications (Kong and Li (2012a), Shen et al. (2015), Liu et al. (2012)) and are available online to the research community.

- **LABELME:** 22,019 images represented as 512 dimensional GIST descriptors (Torralba et al. (2008); Russell et al. (2008))[1] The dataset is mean centred.
- **CIFAR-10:** 60,000 32×32 colour images sampled from the 80 million Tiny Images dataset (Krizhevsky and Hinton (2009)). Each image is encoded with a 512 dimensional GIST descriptor (Oliva and Torralba (2001)) and is manually assigned a label from a selection of 10 classes[2]. Each class has 6,000 associated images. The visual feature descriptors are mean centered.
- **NUS-WIDE:** 269,648 images downloaded from Flickr each annotated with multiple ground truth concept tags (e.g. nature, dog, animal, swimming, car) from

[1] `http://www.cs.toronto.edu/~norouzi/research/mlh/`
[2] `http://www.cs.toronto.edu/~kriz/cifar.html`

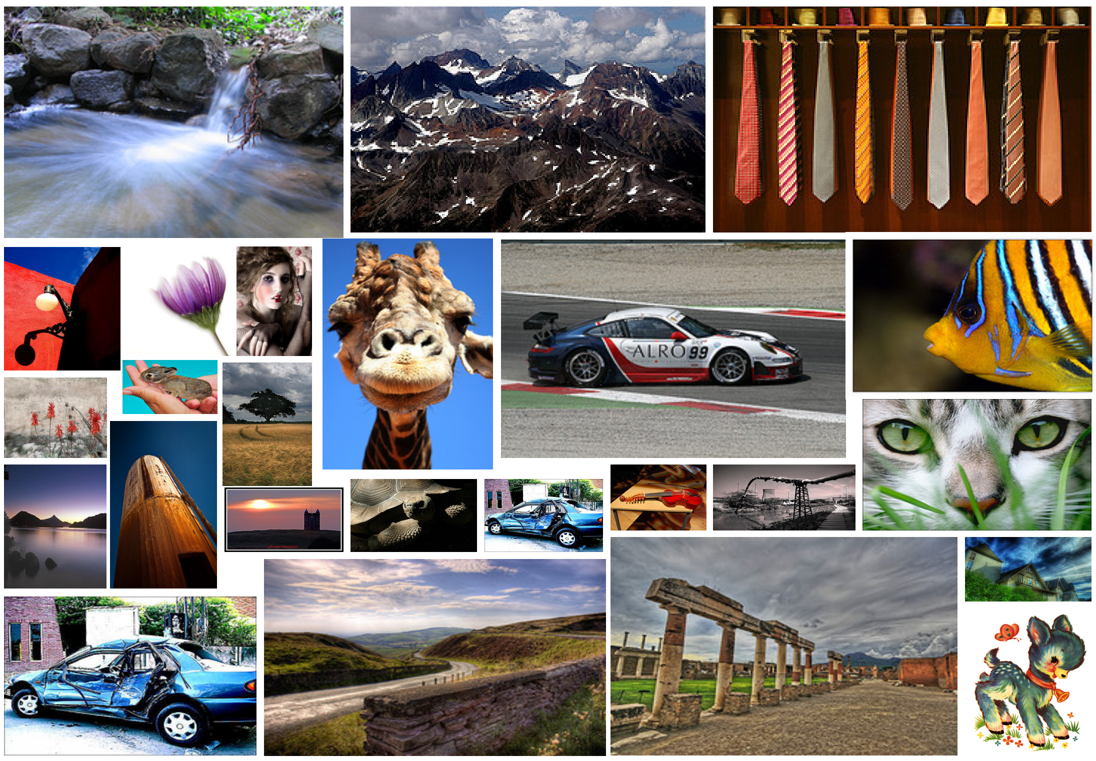


Figure 3.1: The NUS-WIDE dataset consists of around 270,000 images randomly sampled from Flickr. Given the diversity of images (from people, animals, landscapes to buildings and drawings) and widely varying resolution the dataset provides a challenging testbed for image retrieval.

an 81 concept vocabulary[3] (Chua et al. (2009)). The images are represented by a 500 dimensional bag-of-visual-words (BoW) feature descriptor formed by vector quantising SIFT descriptors via k-means clustering. The visual feature descriptors are L_2-normalised to unit length and mean centered. For illustrative purposes I show a random sampling of images from the NUS-WIDE dataset in Figure 3.1.

- **SIFT1M:** 1,000,000 images from Flickr encoded with 128-dimensional SIFT descriptors[4]. This dataset was first introduced by Jegou et al. (2011) and has since became a standard image collection for evaluating nearest neighbour search methods (Kong and Li (2012a), He et al. (2013), Wang et al. (2010b)). The dataset is mean centred.

[3] `http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm`
[4] `http://lear.inrialpes.fr/~jegou/data.php`

Dataset	# images	# labels	Labels/image	Images/label	Descriptor
LABELME	22,019	–	–	–	512-D Gist
CIFAR-10	60,000	10	1	6,000	512-D Gist
NUS-WIDE	269,648	81	1.87	6,220	500-D BoW
SIFT1M	1,000,000	–	–	–	128-D SIFT

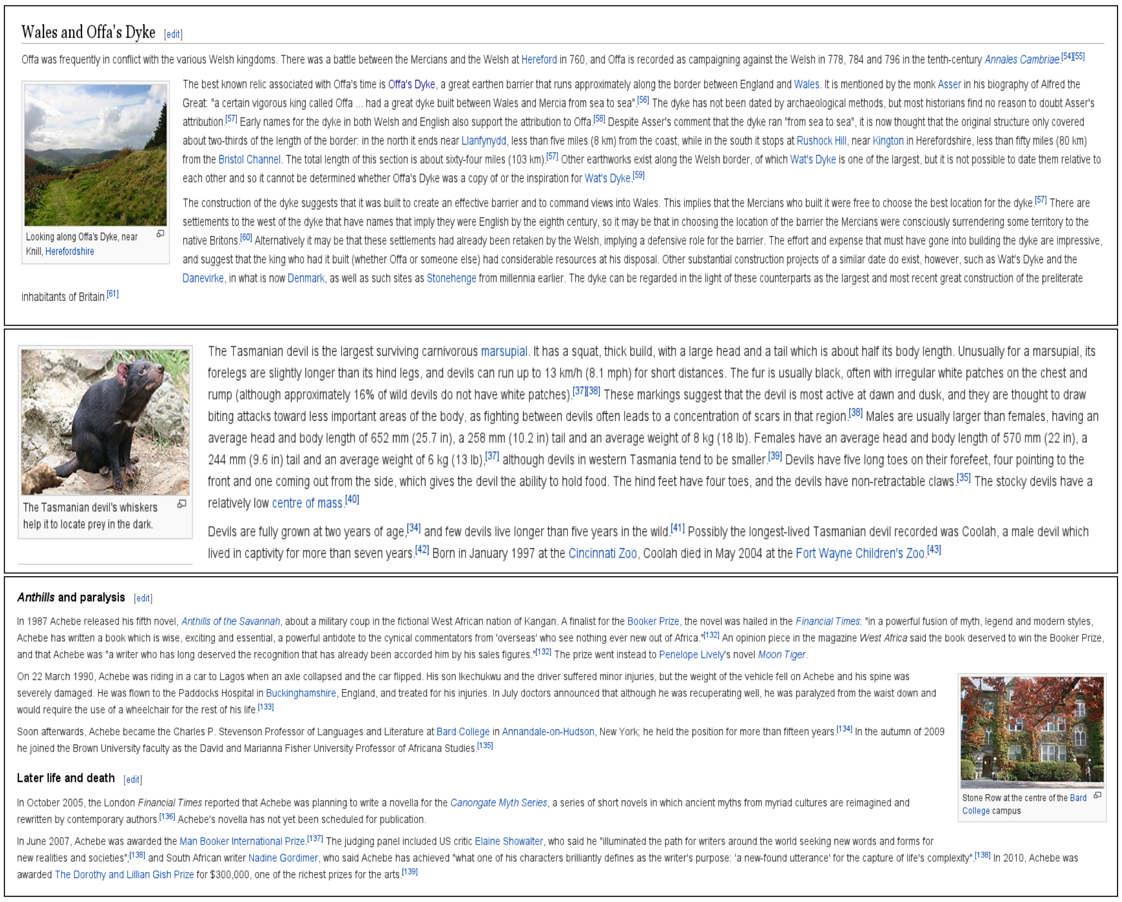
Table 3.1: Salient statistics of the four datasets used in my unimodal experimental evaluation. The Labels/image and Images/label are the mean values computed on the entire dataset.

3.2.2 Cross-modal Retrieval Experiments

The cross-modal retrieval experiments are conducted on the two most popular cross-modal datasets in the learning to hash literature, namely the ‘Wiki’ dataset and NUS-WIDE (Kumar and Udupa (2011); Zhen and Yeung (2012); Song et al. (2013); Rastegari et al. (2013); Bronstein et al. (2010)). Both datasets come with images and associated paired textual descriptors, a key requirement for training and evaluating a cross-modal retrieval model. As for the unimodal retrieval datasets described in Section 3.2.1 these two cross-modal datasets are also freely available to the research community.

- **Wiki:** is generated from 2,866 Wikipedia articles[5] derived from Wikipedia’s “feature articles” (Rasiwasia et al. (2010)). The featured articles segment of Wikipedia hosts the highest quality articles on the site as judged by a panel of independent Wikipedia editors. Each feature article is a document consisting of multiple sections and annotated with at least one relevant image from the Wikimedia commons. Each article is designated with a manually labelled category out of 29 possibilities. Rasiwasia et al. (2010) only keep the articles pertaining to the 10 most populated categories. Each article is further split by section and the image manually placed in that section by the author(s) is used as the corresponding visual description of the text in that section. Any section that ends up without an associated image is discarded. This leaves 2,866 short and focused “articles” of a median length of 200 words, with each article having at least 70 words. I use the image and text feature set provided by Rasiwasia et al. (2010) which is used in most related cross-modal hashing research (Zhen and Yeung (2012)). The visual modality is represented as a 128-dimensional SIFT (Lowe

[5]`http://www.svcl.ucsd.edu/projects/crossmodal/`



Wales and Offa's Dyke [edit]

Offa was frequently in conflict with the various Welsh kingdoms. There was a battle between the Mercians and the Welsh at Hereford in 760, and Offa is recorded as campaigning against the Welsh in 778, 784 and 796 in the tenth-century *Annales Cambriae*.[54][55]

Looking along Offa's Dyke, near Knill, Herefordshire

The best known relic associated with Offa's time is Offa's Dyke, a great earthen barrier that runs approximately along the border between England and Wales. It is mentioned by the monk Asser in his biography of Alfred the Great: "a certain vigorous king called Offa ... had a great dyke built between Wales and Mercia from sea to sea".[56] The dyke has not been dated by archaeological methods, but most historians find no reason to doubt Asser's attribution.[57] Early names for the dyke in both Welsh and English also support the attribution to Offa.[58] Despite Asser's comment that the dyke ran "from sea to sea", it is now thought that the original structure only covered about two-thirds of the length of the border: in the north it ends near Llanfynydd, less than five miles (8 km) from the coast, while in the south it stops at Rushock Hill, near Kington in Herefordshire, less than fifty miles (80 km) from the Bristol Channel. The total length of this section is about sixty-four miles (103 km).[57] Other earthworks exist along the Welsh border, of which Wat's Dyke is one of the largest, but it is not possible to date them relative to each other and so it cannot be determined whether Offa's Dyke was a copy of or the inspiration for Wat's Dyke.[59]

The construction of the dyke suggests that it was built to create an effective barrier and to command views into Wales. This implies that the Mercians who built it were free to choose the best location for the dyke.[57] There are settlements to the west of the dyke that have names that imply they were English by the eighth century, so it may be that in choosing the location of the barrier the Mercians were consciously surrendering some territory to the native Britons.[60] Alternatively it may be that these settlements had already been retaken by the Welsh, implying a defensive role for the barrier. The effort and expense that must have gone into building the dyke are impressive, and suggest that the king who had it built (whether Offa or someone else) had considerable resources at his disposal. Other substantial construction projects of a similar date do exist, however, such as Wat's Dyke and the Danevirke, in what is now Denmark, as well as such sites as Stonehenge from millennia earlier. The dyke can be regarded in the light of these counterparts as the largest and most recent great construction of the preliterate inhabitants of Britain.[61]

The Tasmanian devil's whiskers help it to locate prey in the dark.

The Tasmanian devil is the largest surviving carnivorous marsupial. It has a squat, thick build, with a large head and a tail which is about half its body length. Unusually for a marsupial, its forelegs are slightly longer than its hind legs, and devils can run up to 13 km/h (8.1 mph) for short distances. The fur is usually black, often with irregular white patches on the chest and rump (although approximately 16% of wild devils do not have white patches).[37][38] These markings suggest that the devil is most active at dawn and dusk, and they are thought to draw biting attacks toward less important areas of the body, as fighting between devils often leads to a concentration of scars in that region.[38] Males are usually larger than females, having an average head and body length of 652 mm (25.7 in), a 258 mm (10.2 in) tail and an average weight of 8 kg (18 lb). Females have an average head and body length of 570 mm (22 in), a 244 mm (9.6 in) tail and an average weight of 6 kg (13 lb),[37] although devils in western Tasmania tend to be smaller.[39] Devils have five long toes on their forefeet, four pointing to the front and one coming out from the side, which gives the devil the ability to hold food. The hind feet have four toes, and the devils have non-retractable claws.[35] The stocky devils have a relatively low centre of mass.[40]

Devils are fully grown at two years of age,[34] and few devils live longer than five years in the wild.[41] Possibly the longest-lived Tasmanian devil recorded was Coolah, a male devil which lived in captivity for more than seven years.[42] Born in January 1997 at the Cincinnati Zoo, Coolah died in May 2004 at the Fort Wayne Children's Zoo.[43]

***Anthills* and paralysis** [edit]

In 1987 Achebe released his fifth novel, *Anthills of the Savannah*, about a military coup in the fictional West African nation of Kangan. A finalist for the Booker Prize, the novel was hailed in the *Financial Times*: "in a powerful fusion of myth, legend and modern styles, Achebe has written a book which is wise, exciting and essential, a powerful antidote to the cynical commentators from 'overseas' who see nothing ever new out of Africa."[132] An opinion piece in the magazine *West Africa* said the book deserved to win the Booker Prize, and that Achebe was "a writer who has long deserved the recognition that has already been accorded him by his sales figures."[132] The prize went instead to Penelope Lively's novel *Moon Tiger*.

On 22 March 1990, Achebe was riding in a car to Lagos when an axle collapsed and the car flipped. His son Ikechukwu and the driver suffered minor injuries, but the weight of the vehicle fell on Achebe and his spine was severely damaged. He was flown to the Paddocks Hospital in Buckinghamshire, England, and treated for his injuries. In July doctors announced that although he was recuperating well, he was paralyzed from the waist down and would require the use of a wheelchair for the rest of his life.[133]

Soon afterwards, Achebe became the Charles P. Stevenson Professor of Languages and Literature at Bard College in Annandale-on-Hudson, New York; he held the position for more than fifteen years.[134] In the autumn of 2009 he joined the Brown University faculty as the David and Marianna Fisher University Professor of Africana Studies.[135]

Stone Row at the centre of the Bard College campus

Later life and death [edit]

In October 2005, the London *Financial Times* reported that Achebe was planning to write a novella for the *Canongate Myth Series*, a series of short novels in which ancient myths from myriad cultures are reimagined and rewritten by contemporary authors.[136] Achebe's novella has not yet been scheduled for publication.

In June 2007, Achebe was awarded the Man Booker International Prize.[137] The judging panel included US critic Elaine Showalter, who said he "illuminated the path for writers around the world seeking new words and forms for new realities and societies";[138] and South African writer Nadine Gordimer, who said Achebe has achieved "what one of his characters brilliantly defines as the writer's purpose: 'a new-found utterance' for the capture of life's complexity".[138] In 2010, Achebe was awarded The Dorothy and Lillian Gish Prize for $300,000, one of the richest prizes for the arts.[139]

Figure 3.2: Three example Wikipedia article sections (Offa King of Mercia (`https://en.wikipedia.org/wiki/Offa_of_Mercia`), the Tasmanian devil (`https://en.wikipedia.org/wiki/Tasmanian_devil`) and a description from the life of Nigerian novelist Chinua Achebe (`https://en.wikipedia.org/wiki/Chinua_Achebe`) and their aligned images taken from the cross-modal Wiki dataset.

(2004)) bag-of-words histogram, while the textual modality is represented as 10-dimensional probability distribution over Latent Dirichlet Allocation (LDA) topics (Blei et al. (2003)). Three example Wikipedia article sections and their associated images are shown in Figure 3.2.

- **NUS-WIDE:** is identical to the unprocessed NUS-WIDE dataset described in Section 3.2.1 (Chua et al. (2009)). For my cross-modal experiments I pre-process the dataset in a different manner to the strategy described in Section 3.2.1 so that my experiments are compatible with those presented in the relevant literature (Zhen and Yeung (2012)). More specifically, for cross-modal retrieval the multiple image tags associated with an image are used to define the textual modality. I keep the image-text pairs associated with the most frequent 10 classes. Each image is associated with a subset of 5,018 tags manually assigned by Flickr users. I perform a PCA dimensionality reduction on the $269{,}648 \times 5018$ dimensional tag co-occurrence matrix to form a 1,000-dimensional tag feature set. This projected tag feature set is then mean-centered and used as a representation of the textual modality. This is a standard pre-processing step in the literature (Zhen and Yeung (2012)). The visual modality is represented by the same 500 dimensional bag-of-words (BoW) feature descriptors described in the context of NUS-WIDE in Section 3.2.1. The visual descriptors are L_2-normalised to unit length and mean centered.

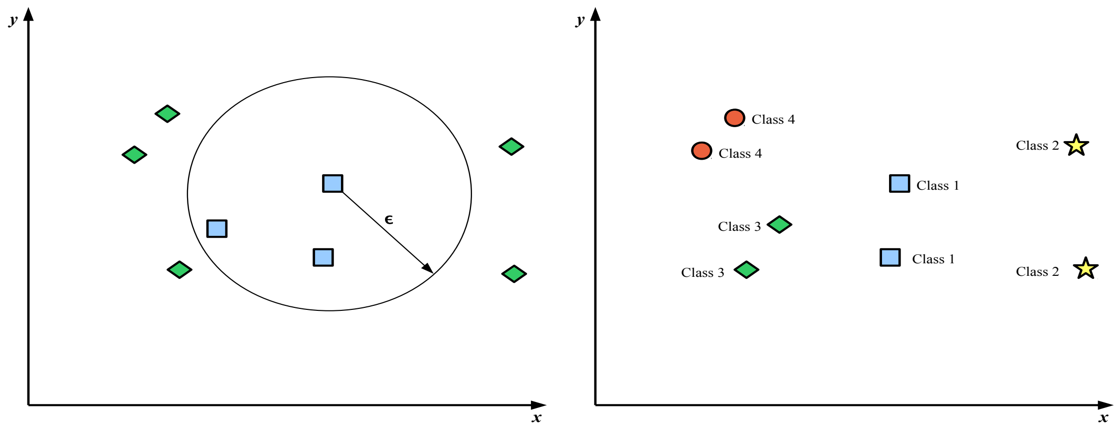
3.3 Nearest Neighbour Groundtruth Definition

In order to evaluate the retrieval effectiveness of a hashing model we need to define which data-points in the database are considered to be nearest neighbours of the query data-points. I refer to these data-points as the *true* nearest neighbours of a query. The system is penalised depending on the degree to which it fails to return the true nearest neighbours for a query. The definition of the groundtruth nearest neighbours varies widely between publications. In this thesis I consider two of the strategies commonly used to define groundtruth which involves either constructing an ε-ball around the query data-point (Section 3.3.1) or using human assigned class-labels (Section 3.3.2). To the best of my knowledge, there has been no work to verify whether or not the ε-ball groundtruth definition correlates with user search satisfaction. I discuss this point further in Chapter 8 as part of possible future work.

3.3.1 ε-Ball Nearest Neighbours

I opt primarily for the ε-nearest neighbour (ε-NN) definition in this thesis (Figure 3.3a)[6]. In this paradigm a ball of radius ε is defined around a query data-point in the input feature space and the true nearest neighbours are defined as those data-points enclosed within the ball. To compute the ε-NN groundtruth I follow previous related work (Kong et al. (2012); Kong and Li (2012a); Kulis and Darrell (2009); Gong and Lazebnik (2011)) and randomly sample 100 data-points from the training dataset to

[6]Defining ground-truth nearest neighbours can also be achieved by computing a k-NN graph. In this case related data-points to a query are those that have the k smallest distances to the query. I leave this type of evaluation as future work.



(a) ε-**nearest neighbour ground truth** (b) **Class based groundtruth**

Figure 3.3: Two definitions of groundtruth nearest neighbours (NN). In Figure (a) I show how to define nearest neighbours of a data-point using an ε-ball. All data-points enclosed by the ball are nearest neighbours of the data-point at the center. In Figure (b) I show a class-based definition of nearest neighbours. Nearest neighbours are defined as those data-points sharing at least one class label in common.

compute the Euclidean distance at which each data-point has R nearest neighbours on average. The ε-ball radius is then set to equal this average distance. The parameter R is set to 50 nearest neighbours in the literature (Kong et al. (2012); Kong and Li (2012a,b); Kulis and Darrell (2009); Raginsky and Lazebnik (2009); Gong and Lazebnik (2011)), and for compatibility I use the same setting throughout this thesis. The groundtruth matrix $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ is then derived by computing the Euclidean distance $\mathbf{D} \in \mathbb{R}^{N_{trd} \times N_{trd}}$ between a small subset of the data-points ($N_{trd} \ll N$) and thresholding the distances by ε. This method of groundtruth generation is presented in Equation 3.1 and Figure 3.3a.

$$\mathbf{S} = \begin{cases} S_{ij} = 1, & \text{if} \quad D_{ij} \leq \varepsilon \\ S_{ij} = 0, & \text{if} \quad D_{ij} > \varepsilon \end{cases} \tag{3.1}$$

3.3.2 Class-Based Nearest Neighbours

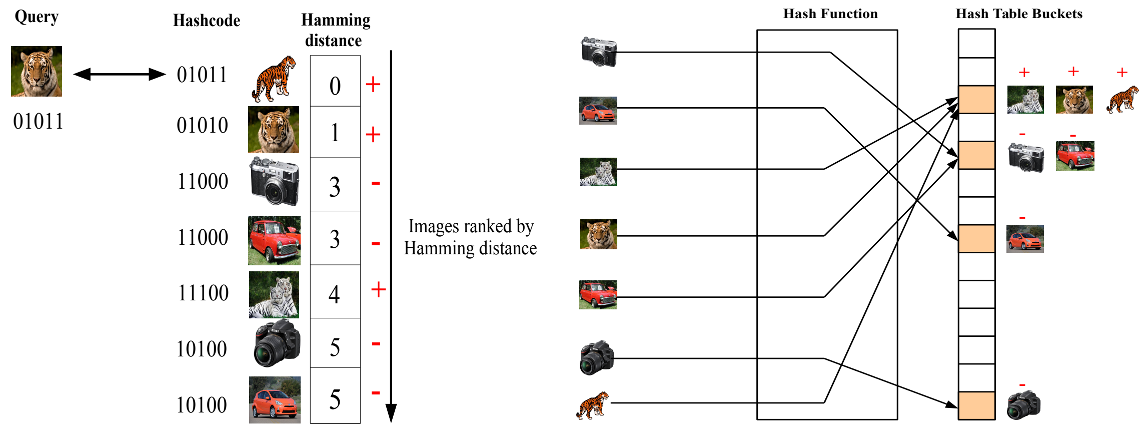
Experimental results will be presented based on class labelled derived groundtruth (Figure 3.3b) in situations where the ε-NN evaluation paradigm is not possible such as for cross-modal retrieval (it is not possible to directly compute the Euclidean distance between two feature vectors of a different type) or where I wish to present additional results that can be directly compared to a specific portion of the literature that traditionally only uses a class-label based evaluation (e.g. the data-dependent supervised models discussed in Section 2.6.4). In this scenario $S_{ij} = 1$ for an element of the groundtruth matrix $\mathbf{S}$ if the corresponding pair of data-points $\mathbf{x}_i, \mathbf{x}_j$ share *at least one* class label or annotation in common, and $S_{ij} = 0$ otherwise. This is the same strategy used by most related research in the learning to hash literature (Gong and Lazebnik (2011); Liu et al. (2012)).

3.4 Evaluation Paradigms

There are two main paradigms for evaluating hashing models: the *Hamming ranking* evaluation paradigm (Section 3.4.1) and the *hashtable bucket* evaluation paradigm (Section 3.4.2). Both paradigms are illustrated in Figure 3.4. The Hamming ranking paradigm is standard within the learning to hash research literature while the hashtable bucket evaluation paradigm is frequently used in practical hashing applications in which a fast query-time is of prime importance. I introduce both paradigms in Sections 3.4.1-3.4.2 before explaining why I use the Hamming ranking evaluation paradigm exclusively throughout this thesis in Section 3.4.3.

3.4.1 Hamming Ranking Evaluation

In all the experiments in this thesis I will follow previous related research (Kong et al. (2012); Kong and Li (2012a,b); Liu et al. (2012, 2011); Gong and Lazebnik (2011); Zhang et al. (2010b); Kulis and Grauman (2009)) and evaluate retrieval effectiveness using the widely accepted *Hamming ranking evaluation* paradigm. In this evaluation paradigm, binary hashcodes are generated for both the query and the database images. The Hamming distance is then computed from the query images to all of the database images, with the database dataset images ranked in ascending order of the Hamming distance. The resulting ranked lists are then used to compute retrieval evaluation metrics such as area under the precision recall curve (AUPRC) (Section 3.6.3) and mean average precision (mAP) (Section 3.6.4). The Hamming ranking evaluation paradigm is a proxy for evaluating hashing accuracy over the range of user preferences (precision/recall) and without having to specify the parameters (K, L) of a specific hashtable implementation. I discuss this latter point further in Section 3.4.3.



(a) **Hamming ranking evaluation** (b) **Hashtable bucket evaluation**

Figure 3.4: Two evaluation paradigms for hashing. In both cases relevant images are those depicting similar objects. In Figure (a) the Hamming distance between the query hashcode and the database images is computed. The images are ranked and the resulting ranked list used to compute a retrieval metric such as average precision (AP). In this case we find an $AP = 0.87$. The average precision scores are aggregated across queries by computing the mean average precision (mAP). In Figure (b) I show the hashtable evaluation strategy. Each image is hashed to a bucket. A count is then made of the number of true positives (TPs), false positives (FPs) and false negatives (FNs) colliding in the same buckets. In this toy example, $TP = 3$, $FP = 1$, $FN = 2$ which equates to a micro-average F_1-measure of 0.78. On both diagrams $+$ indicates a true positive while a $-$ indicates a false positive/negative.

3.4.2 Hashtable Bucket-Based Evaluation

The Hamming ranking evaluation paradigm is by definition of $O(N)$ time complexity for a single query data-point. The *hash bucket-based evaluation* has a constant $O(1)$ search time independent of the dataset size. A hashtable lookup evaluation is much closer to how the hashing models would be used in a real-world application where a fast query time is a necessity. Despite this fact a hashtable evaluation is rarely reported in the learning to hash literature with the Hamming ranking paradigm being the preferred evaluation methodology. I previously described the application of hashtables in the context of Locality Sensitive Hashing (LSH) (Chapter 2, Section 2.4). In this evaluation paradigm the hashcodes are generated for the query and database points which are then used as the indices into the buckets of L hashtables. The union is then taken over all the data-points that collide in the same buckets as the query across the

L hashtables. The set of data-points thus formed can then be used to compute the effectiveness metrics of precision, recall and F_β-measure. I describe these metrics in more detail in Section 3.6, but the intuition is that we want to reward the algorithm if it returns many true nearest neighbours to the query in the retrieved set while penalising it for returning unrelated data-points. As we examine all colliding data-points for a query we are therefore using the second hashtable query strategy which was discussed in the description of LSH in Chapter 2, Section 2.4.

3.4.3 Hamming Ranking versus Hashtable Bucket Evaluation

The hashtable bucket evaluation paradigm is heavily dependent on both the particular hashtable implementation (i.e. values of K, L, whether or not chaining is used, etc) and on the end application itself, for example is the hashtable on a drone and therefore do we have limited available main memory? If I opt for a hashtable evaluation paradigm I either need to pick a default setting of K and L and tie my evaluation to this specific hashtable implementation, or alternatively I can measure the hashing model performance over many different values of the hashtable parameters leading to an explosion in the number of results to be reported. To abstract away from the specifics of a particular hashtable implementation and to obtain a single number summarising the quality of the hashcodes, researchers in the Computer Vision literature prefer to evaluate their hashing models by Hamming ranking which involves computing the *Hamming distance* between the hashcodes, rather than using a hashtable-based setup (Gong and Lazebnik (2011), Liu et al. (2011)). The Hamming distance given in Equation 2.4 (Chapter 2, Section 2.3) measures the number of bits that are different between two hashcodes and is therefore likely to be a good indicator of the quality of a hashtable lookup using those hashcodes. The more bits in common the greater the likelihood of a collision between the corresponding data-points.

There is an interesting, but not immediately obvious link between the Hamming ranking evaluation paradigm and the hashtable evaluation paradigm. Measuring the quality of a set of ranked lists using mAP and AUPRC is effectively acting as a *proxy* for *many* different settings of K and L in a corresponding hashtable evaluation. To confirm this fact, I make reference to Figure 3.5 in which I show five hashcodes ranked in ascending order of Hamming distance from the query (marked in the diagram in bold font). Observe that each threshold effectively defines a set of "colliding" data-points, that is those above the threshold with the lowest Hamming distance to the query. The

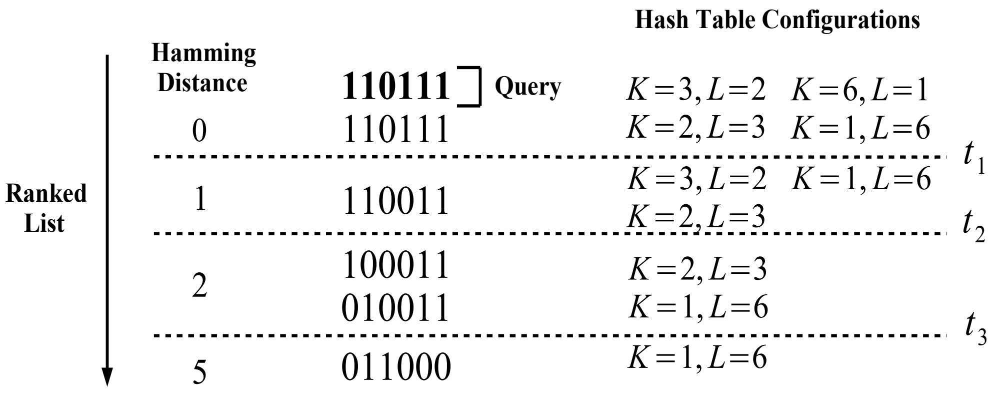

Hash Table Configurations
Hamming Distance
Ranked List
110111 Query
$K=3,L=2$ $K=6,L=1$
0 110111 $K=2,L=3$ $K=1,L=6$
t_1
1 110011 $K=3,L=2$ $K=1,L=6$ $K=2,L=3$
t_2
2 100011 010011 $K=2,L=3$ $K=1,L=6$
t_3
5 011000 $K=1,L=6$

Figure 3.5: An analogy between the Hamming ranking evaluation paradigm and the hashtable bucket evaluation paradigm. I rank five hashcodes in ascending order by Hamming distance from the query hashcode (shown here in bold). The ranked list is thresholded at *three* different points (t_1, t_2, t_3), with the thresholds indicated by the dashed horizontal lines. The hashcodes above the threshold can be considered to be "colliding" with the query hashcode. The settings of K and L that would cause the collision are shown for the four different Hamming distances. For example, for the *bottom most* thresholding splitting the hashcodes into $L = 6$ segments of $K = 1$ bits will cause the hashcode at Hamming distance 5 to collide in the same bucket as the hashcodes at Hamming distance 2,1 and 0. In this way we see that a particular thresholding of a ranked list of hashcodes is equivalent to many different settings of K and L in a hashtable evaluation.

thresholded ranked list therefore corresponds to settings of K and L that ensure the data-points above the threshold will collide in at least one hypothetical hashtable. The L hashtables in Figure 3.5 are formed by splitting the hashcodes into L K-bit segments, with each K-bit segment indexing into a specific bucket of one of the L hashtables. Just as choosing a particular setting of K and L is application specific so is choosing a particular threshold in the Hamming ranking evaluation paradigm. Usefully the mAP and AUPRC provide a single number measure of ranking quality that is computed by aggregating across many different settings of the ranked list threshold, and consequently many different values of K and L. The Hamming ranking evaluation paradigm is therefore a more general evaluation strategy for hashing that is able to measure the overall quality of hashcodes without being tied to a particular end-application. In effect it indicates how good the hashing-based ANN search would be if we found the best setting of K and L in a hashtable bucket evaluation. Given this attractive advantage I accord with the relevant literature and follow the Hamming ranking evaluation strategy throughout this thesis (Chapters 4-7). I leave a hashtable bucket evaluation to future work.

3.5 Constructing Random Dataset Splits

In this section I will describe how the datasets introduced in Section 3.2 are partitioned to form testing and validation queries and training and database splits for the purposes of learning and evaluating the hash functions. Two strategies for forming splits will be described: in Section 3.5.1 I describe the literature standard strategy that is widely used by the research community, while in Section 3.5.2 I describe my proposed splitting methodology that seeks to remedy concerns with the accepted evaluation strategy. In both cases, when class-based ground-truth is used, I sample the splits so as to obtain a balanced distribution of classes within each partition. This sampling strategy is discussed in more detail in Chapter 6, Section 6.3.

3.5.1 Literature Standard Splits

In previous work *repeated random subsampling* cross-validation over ten independent runs is used to evaluate the quality of the learnt hash functions (Liu et al. (2012); Kong et al. (2012); Kong and Li (2012a); Liu et al. (2014); Wang et al. (2012)). Figure 3.6 shows an example of a random dataset split for one run. The entire dataset is denoted as $\mathbf{X} \in \mathbb{R}^{N \times D}$. This dataset is divided into a held-out set of test queries $\mathbf{X}_{teq} \in \mathbb{R}^{N_{teq} \times D}$ and a database split $\mathbf{X}_{db} \in \mathbb{R}^{N_{db} \times D}$. The test queries are used once when I come to compute the evaluation metric by ranking the database split (Section 3.4.1). The database split also doubles as the training dataset for learning the hash functions. The best setting of model *hyperparameters*[7] is found by grid search on the validation split of the dataset. In practice this grid search is conducted by running a set of validation queries $\mathbf{X}_{vaq} \in \mathbb{R}^{N_{vaq} \times D}$ against a validation database $\mathbf{X}_{vad} \in \mathbb{R}^{N_{vad} \times D}$, both of which are sampled from the database $\mathbf{X}_{db}$. I am therefore using a form of nested cross-validation in which the optimal hyperparameters are determined for each run. The training database $\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D}$ is used to learn the *parameters* (hyperplanes, quantisation thresholds) of the hash functions and is itself a subset of $\mathbf{X}_{db}$. In the remainder of this dissertation I refer to this splitting strategy as the *literature standard splitting strategy*. I illustrate this method of forming dataset splits in Figure 3.6.

[7]Hyperparameters are parameters *other* than the hashing hyperplanes or quantisation thresholds. Example of hyperparameters are the flexibility of margin C for the SVM and the kernel bandwidth parameter γ for the RBF kernel.

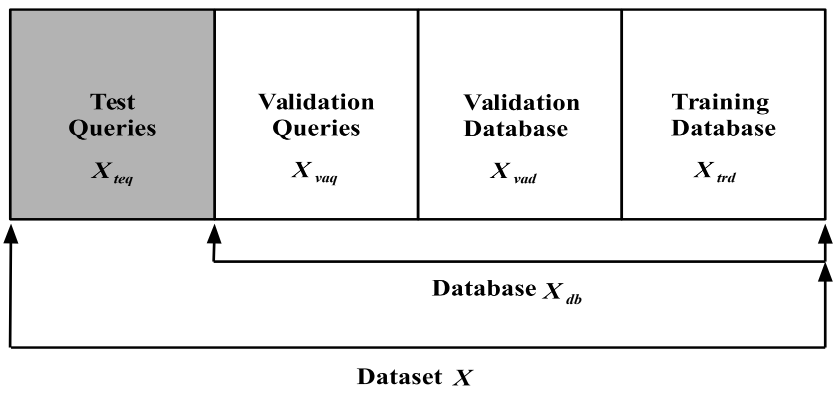

Test Queries X_{teq}
Validation Queries X_{vaq}
Validation Database X_{vad}
Training Database X_{trd}
Database X_{db}
Dataset X

Figure 3.6: The literature standard dataset splitting procedure. The standard procedure used in the literature for splitting a dataset into testing and training partitions. The entire dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$ is represented as the concatenation of the individual rectangles, each of which highlights a particular partition. The rectangle in grey represents the split of the dataset that is held-out and only used once for computing the final measure of retrieval effectiveness.

3.5.2 Improved Splitting Strategy

Unfortunately, there is a potential overfitting concern with the standard dataset splitting strategy described in Section 3.5.1 given that the database points which are ranked or indexed with respect to the test queries are also used as the training dataset for learning the hash functions themselves. Ideally there should be a clean separation between the split of the dataset that is used to learn the hash functions and the split of the dataset that is ranked/indexed in order to compute the final measure of retrieval effectiveness. This ensures that I can evaluate the true generalisation performance of the hash functions when there is not only unseen queries but also an unseen database that is to be ranked/indexed with respect to those queries. Currently the literature is only concerned with the generalisation performance with respect to unseen query data-points and where the database is known a-priori and can be used for hash function learning. To the best of my knowledge I am the first in the literature to note this technical flaw in the standard method for forming dataset splits. To mitigate this overfitting concern I propose a new method for generating splits of the dataset. In this new strategy I again perform repeated random subsampling cross-validation over ten runs. However, the makeup of a random split for a run now differs from the literature standard splitting strategy. In my suggested dataset splitting strategy I divide the dataset into *five* splits as shown in Figure 3.7. I have a set of held-out test queries $\mathbf{X}_{teq} \in \mathbb{R}^{N_{teq} \times D}$ and also a

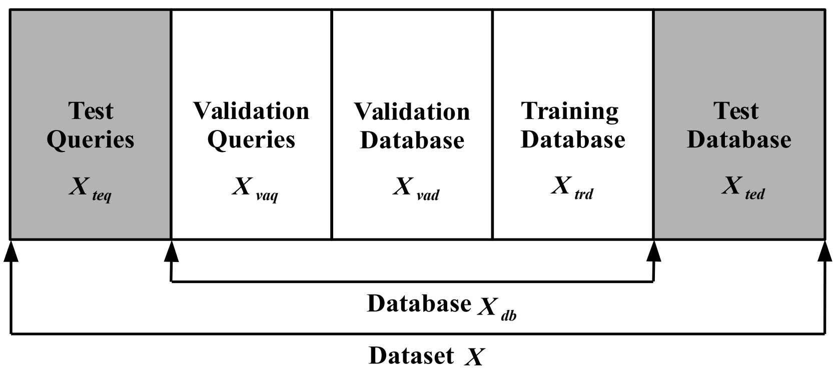

Test Queries X_{teq}
Validation Queries X_{vaq}
Validation Database X_{vad}
Training Database X_{trd}
Test Database X_{ted}
Database X_{db}
Dataset X

Figure 3.7: My improved dataset splitting procedure. My proposed splitting strategy for overcoming the overfitting concern with the literature standard strategy. In addition to the test queries I also advocate holding out a split of the dataset to act as the testing database. The held-out splits of the dataset are shown in grey. At test time the testing queries are used to retrieve related items from the testing database. This retrieval run is used to compute the final measure of effectiveness for determining the quality of the hash functions.

held-out test database $\mathbf{X}_{ted} \in \mathbb{R}^{N_{ted} \times D}$ against which those test queries are run. Both of the test queries and test database are only used once when I come to compute the final retrieval effectiveness metric for that particular run. The remainder of the dataset forms the database split $\mathbf{X}_{db} \in \mathbb{R}^{N_{db} \times D}$ which is used for setting the parameters and hyperparameters of the hashing models. The database split is further divided into a set of validation queries $\mathbf{X}_{vaq} \in \mathbb{R}^{N_{vaq} \times D}$, a validation database split $\mathbf{X}_{vad} \in \mathbb{R}^{N_{vad} \times D}$ which is ranked/indexed against the validation queries and a training split that is used to learn the hash functions $\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D}$. For the remainder of this dissertation I refer to this splitting strategy as the *improved splitting strategy*.

3.6 Evaluation Metrics

I follow previous research in the learning to hash literature and judge the retrieval effectiveness by the standard Information Retrieval (IR) metrics of *precision*, *recall*, *F_β-measure* (Section 3.6.1), *area under the precision recall curve (AUPRC)* (Section 3.6.3) and *mean average precision (mAP)* (Section 3.6.4).

3.6.1 Precision, Recall, F_β-Measure

Precision, *recall* and their harmonic mean, the F_β-*measure*, are set-based evaluation metrics that can be used to ascertain the quality of an unranked collection of images. The retrieved set can be determined by looking into the colliding hashtable buckets for the Hashtable bucket evaluation or by defining a Hamming radius threshold for the Hamming ranking evaluation. In terms of the Hamming ranking evaluation, precision and recall can be computed by counting the number of true nearest neighbours that are within a fixed Hamming radius (true positives, TPs), the number of non nearest neighbours that are within a fixed Hamming radius (false positives, FPs) and the number of related data-points that are *not* are within a fixed Hamming radius to the query (false negatives, FNs).

More formally, I denote the groundtruth matrix as $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ (Sections 3.3.1-3.3.2). The groundtruth adjacency matrix specifies which data-points are *true nearest neighbour* pairs ($S_{ij} = 1$) and which data-point pairs are unrelated ($S_{ij} = 0$). As we discussed in Section 3.3, in the context of hashing-based ANN search, a data-point $\mathbf{x}_j$ is denoted as a true nearest neighbour ($S_{ij} = 1$) if it is within an ε-ball of the query data-point $\mathbf{q}_i$ or shares at least one class label in common with the query. Following a retrieval run, the ranked data-points within a certain Hamming radius (D) of the query are those data-points considered to be related to the query, while those data-points outside of the Hamming radius D are considered to be unrelated[8]. The results of a ranked retrieval for a certain Hamming distance threshold D are represented by the square matrix $\mathbf{R} \in \{0,1\}^{N \times N}$ given in Equation 3.2.

$$R_{ij} = \begin{cases} 1, & \text{if} \quad \mathbf{x}_j \quad \text{is within Hamming radius D to the query} \quad \mathbf{q}_i \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

Given the definitions of $\mathbf{S}$ and $\mathbf{R}$, the number of *true positives* for a single query data-point $\mathbf{q}_i$ is defined in Equation 3.3

$$TP(\mathbf{q}_i) = \sum_j S_{ij} \cdot R_{ij} \tag{3.3}$$

A *false negative (FN)* is a true nearest neighbour ($S_{ij} = 1$) that is outside of the Hamming radius around the query $\mathbf{q}_i \in \mathbb{R}^D$. The total false negative count for the query is

[8] This is the Hamming ranking evaluation paradigm discussed in Section 3.4.1.

given in Equation 3.4

$$FN(\mathbf{q}_i) = \sum_j S_{ij} - TP(\mathbf{q}_i) \tag{3.4}$$

A *false positive (FP)* is a non-nearest neighbour ($S_{ij} = 0$) that falls within the query Hamming radius $\mathbf{q}_i \in \mathbb{R}^D$ (Equation 3.5)

$$FP(\mathbf{q}_i) = \sum_j (1 - S_{ij}) \cdot R_{ij} \tag{3.5}$$

Given Equations 3.3-3.5 the precision and recall metrics can then be defined as in Equations 3.6-3.7

$$P(\mathbf{q}_i) = \frac{TP(\mathbf{q}_i)}{TP(\mathbf{q}_i) + FP(\mathbf{q}_i)} \tag{3.6}$$

Precision is therefore the fraction of true nearest neighbours that are within the fixed Hamming radius out of all data-points that are within the fixed Hamming radius to the query data-point

$$R(\mathbf{q}_i) = \frac{TP(\mathbf{q}_i)}{TP(\mathbf{q}_i) + FN(\mathbf{q}_i)} \tag{3.7}$$

Recall is then the fraction of true nearest neighbours that are within the fixed Hamming radius to the query out of all possible true nearest neighbours for that query, regardless whether or not they are within the specified Hamming radius.

In a typical image retrieval experiment we have more than one query data-point. The question arises as to how we aggregate the precision and recall scores for all Q queries. There are effectively two ways which involve either taking a *micro-average* or a *macro-average*. To accord with the literature I am most interested in the *micro-average* in this thesis which would sum the TPs, FPs and FNs across all queries before computing the total precision and recall (Equations 3.8-3.9).

$$P_{micro} = \frac{\sum_{i=1}^{Q} TP(\mathbf{q}_i)}{\sum_{i=1}^{Q} TP(\mathbf{q}_i) + \sum_{i=1}^{Q} FP(\mathbf{q}_i)} \tag{3.8}$$

$$R_{micro} = \frac{\sum_{i=1}^{Q} TP(\mathbf{q}_i)}{\sum_{i=1}^{Q} TP(\mathbf{q}_i) + \sum_{i=1}^{Q} FN(\mathbf{q}_i)} \tag{3.9}$$

Finally, the weighted harmonic mean of recall and precision is known as the F_β-measure and is presented in Equation 3.10 (Rijsbergen (1979)):

$$\begin{aligned} F_{\beta} &= \frac{(1+\beta^2)P_{micro}R_{micro}}{\beta^2 P_{micro} + R_{micro}} \\ &= \frac{(1+\beta^2)TP_{micro}}{(1+\beta^2)TP_{micro} + \beta^2 FN_{micro} + FP_{micro}} \end{aligned} \tag{3.10}$$

F_{β}-measure can be used to combine precision and recall resulting from both a macro or micro-average. The free parameter $\beta \in \mathbb{R}_{+}$ is used to adjust the contribution from the precision and recall. Setting $\beta < 1$ in Equation 3.10 weights precision higher than recall, and vice-versa for a setting of $\beta > 1$. In most applications β is set to 1.0 giving the commonly used F_1-measure that provides an equal balance between the contribution of precision and recall to the final score. The greater the F_{β}-measure the more effective are the hash functions at returning true nearest neighbours in the same hashtable buckets.

3.6.2 Precision Recall Curve (PR Curve)

The precision and recall set-based evaluation metrics discussed in Section 3.6.1 are computed at a fixed operating point of the hashing algorithm. This operating point is usually derived from a particular parameter setting that is itself driven by user or system constraints. For example, in the context of a hashtable bucket evaluation paradigm (Section 3.4.2) this threshold could be implicitly defined by varying the number of hashtables L and the number of hashcode bits K. For the Hamming ranking evaluation paradigm (Section 3.4.1) the threshold is the radius of the Hamming ball around the queries. Database points with a Hamming distance to the query that puts them outside of the radius are not considered part of the retrieved set and therefore do not contribute to the computation of the precision and recall metrics. In contrast to the set-based evaluation metrics, the *precision-recall (PR) curve* measures the effectiveness of a ranked list of items across a range of different operating points. For the Hamming ranking evaluation paradigm, the PR curve is constructed by finding all the data-points within a certain Hamming radius D of the query set and computing the precision and recall over the corresponding retrieved set. By varying the Hamming radius from unity to the maximum Hamming radius D_{max} exhibited by database hashcodes we can trace out a PR curve using the resulting D_{max} precision-recall values. This curve depicts the trade-off between precision and recall as the Hamming radius from the queries is gradually increased. We expect that as the Hamming radius is increased the precision

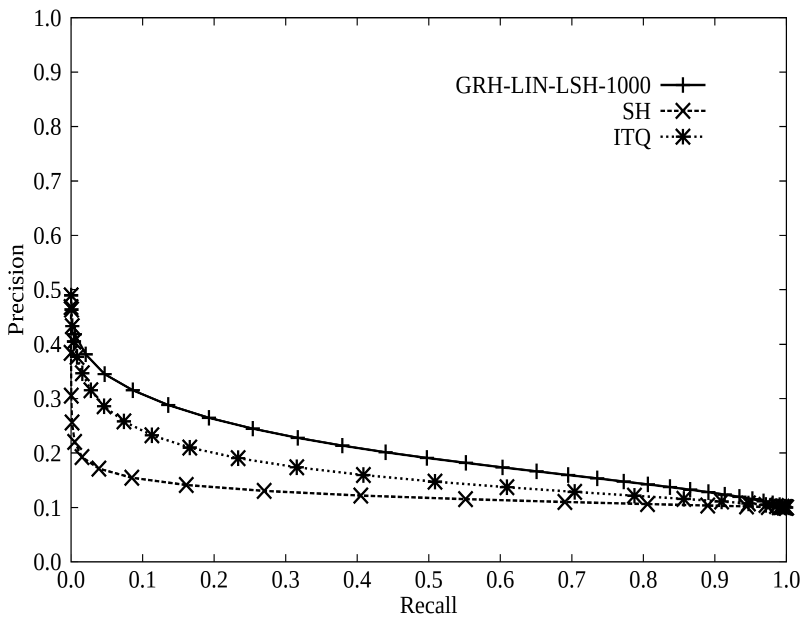


Figure 3.8: Precision recall curves for the CIFAR-10 dataset for a hashcode length of 32 bits. The three hashing algorithms indicated on this graph are studied in Chapter 6.

will drop (as more non-relevant data-points are encountered) while the recall will increase (as more relevant data-points are retrieved). An example precision-recall curve is presented in Figure 3.8.

3.6.3 Area Under the Precision Recall Curve

In many situations a single number summarising the ranking effectiveness captured by the precision-recall curve is required. Given its wide application in previously related research (Kong et al. (2012); Kong and Li (2012a,b); Moran et al. (2013a,b)) I settle for the *area under the precision-recall curve (AUPRC)* as the main single number effectiveness metric used consistently throughout this dissertation. The AUPRC is a real-valued number constrained to be within the limits of 0 and 1 and provides a summary of the retrieval effectiveness across all levels of recall. The computation of AUPRC is defined in Equation 3.11.

$$\begin{aligned} AUPRC &= \int_0^1 P(R)dR \\ &= \sum_{d=1}^{D_{max}} P(d)\delta R(d) \end{aligned} \tag{3.11}$$

where $P(R)$ denotes the micro precision at micro recall R, $P(d)$ is the precision at

Hamming radius d and $\delta R(d)$ is the change in micro recall between Hamming radius $d-1$ and d[9]. The greater the area under the PR curve (AUPRC) the higher the retrieval effectiveness of the associated hashing model. The ideal PR curve has a precision of 1.0 across all recall levels leading to an AUPRC of 1.0.

3.6.4 Mean Average Precision (mAP)

Mean average precision (mAP) is also a commonly applied single-number evaluation metric for summarising the effectiveness of a ranking. However, in contrast to AUPRC which is directly computed from the precision-recall curve, mAP is calculated from the Q ranked lists that are obtained by computing the Hamming distance from every query data-point $\{\mathbf{q}_i \in \mathbb{R}^D\}_{i=1}^{Q}$ to all the database data-points $\{\mathbf{x}_j \in \mathbb{R}^D\}_{j=1}^{N}$. Given a set of Q ranked lists, mAP is defined as follows (Wu et al. (2015)): denote as L the number of true nearest neighbours for query $\mathbf{q}$ among the retrieved data-points, $P_{\mathbf{q}}(r)$ as the precision for query data-point $\mathbf{q}$ when the top r data-points are returned, and $\delta(r)$ as an indicator function which returns '1' when the r^{th} data-point is a true nearest neighbour of the query and '0' otherwise. The average precision (AP) for a single query $\mathbf{q}$ is then given in Equation 3.12 while the average of this quantity across all Q queries, the mean average precision or mAP, is defined in Equation 3.13.

$$AP(\mathbf{q}) = \frac{1}{L}\sum_{r=1}^{R} P_{\mathbf{q}}(r)\delta(r) \tag{3.12}$$

$$mAP = \frac{1}{|Q|}\sum_{i=1}^{Q} AP(\mathbf{q}_i) \tag{3.13}$$

Equation 3.12 computes the precision at each point when a new relevant image is retrieved. The average precision (AP) for a single query $\mathbf{q}$ is then the mean of these precision values. The mAP is then computed by simply taking the mean of the average precisions across all Q queries (Equation 3.13). mAP is a real-valued number between 0.0 and 1.0, with a higher number indicating a more effective ranked retrieval and favours relevant images retrieved at higher (better) ranks. mAP is frequently used as a single-number evaluation metric in certain sub-fields of the learning to hash literature, particularly supervised and unsupervised data-dependent projection (Liu et al. (2011), Liu et al. (2012), Gong and Lazebnik (2011), Zhang et al. (2010b)). When comparing the contributions in this thesis to those particular sub-fields I will also report mAP in

[9]The finite sum representation for the AUPRC can be computed using the trapezoidal rule. This is implemented as the `trapz` function in Matlab.

addition to AUPRC so that my experimental results are directly comparable to previously published research.

3.6.5 Comparing and Contrasting AUPRC and mAP

The application of AUPRC and mAP as an evaluation metric is not consistent across the learning to hash literature, with some sub-fields (particularly binary quantisation) favouring AUPRC while others (such as data-dependent projection) appear to favour mAP. It is well-known that mAP is approximately the average of the AUPRC for a set of queries (Turpin and Scholer (2006)) so it is interesting to briefly consider here the retrieval scenarios where both metrics are expected to be in agreement and when they are likely to differ.

AUPRC is a *micro-average* in which the individual true positives, false positives and false negatives are aggregated across all Q queries for a specific threshold. The total aggregated counts are then used to compute the precision and recall for each possible setting of the threshold. The resulting precision and recall values can then be used to compute the AUPRC as given by Equation 3.11. In contrast the mAP is a *macro-average* which is found by computing the true positives, false positives and resulting precision per query, per relevant document retrieved and then averaging those precision values across all Q queries (Equations 3.12-3.13).

In practice, differences between the mAP and AUPRC will only arise in retrieval applications in which the distribution of relevant documents across queries is skewed. In this scenario the AUPRC will favour models that return more relevant documents from the queries with a larger number of relevant documents to the detriment of those queries that have a smaller number of relevant documents. In contrast the mAP will weight the contribution of every query equally even if many documents are relevant to some queries and very few to other queries. This equal weighting of queries ensures that mAP is insensitive to the performance variation between those queries that have many relevant documents and other queries that have very few relevant documents. To achieve a high mAP score the system must aim to do well across all queries and not just those with many relevant documents.

In a practical scenario, where the distribution of relevant documents per query is highly imbalanced, the choice of summarising the ranking effectiveness with either mAP or AUPRC is application specific (Sebastiani (2002)). In some cases we may be primarily interested in high effectiveness for the queries with a greater number of relevant documents (AUPRC). This may be appropriate for evaluating system orientated tasks in which we wish to quantify how well the system does as a whole in returning pairs of true nearest neighbours (e.g. plagiarism detection). In other cases we may be equally interested in queries with a much smaller number of true positives (mAP). The latter scenario may arise in a user evaluation situation such as web search where the information retrieval system must not be seen to prioritise retrieval effectiveness for one user over another.

3.7 Summary

In this chapter I introduced the evaluation methodology that is commonly employed in the related research literature and which will be used to measure the effectiveness of my own contributions in this thesis. I began in Section 3.2 by outlining a collection of image and document datasets that will be used for my nearest neighbour (NN) search experiments. The datasets were divided into unimodal (image only) and cross-modal datasets (image-document), and were shown to encompass a large variability in the feature descriptors used to encode the images and documents, as well as the type of objects depicted in the images, their resolution and the total number of images (from 22,019 up to 1 million images) per dataset.

The definition of groundtruth is an important facet of any experimental methodology. In Section 3.3, I introduced two main strategies for judging the quality of a nearest neighbour search algorithm. The first strategy constructs a ball of radius ε around a query and any data-points falling within that radius are deemed true nearest neighbours (Section 3.3.1). The second strategy sets true nearest neighbours to be those data-points that share at least one class label in common with the query (Section 3.3.2). The latter groundtruth definition is required for cross-modal retrieval experiments in which the feature descriptors occupy incommensurate feature spaces making an ε-NN evaluation impractical.

In Section 3.4, I then defined the nearest neighbour search strategy to be used in evaluating the quality of the hashcodes. One natural option is to index the database and query images into hashtable buckets and count the number of true nearest neighbours that fall within the same buckets as the query (Section 3.4.2). Surprisingly I discussed how this *hashtable lookup* evaluation strategy is not at all common in the learning to hash literature. Instead most publications of note use what is termed the *Hamming ranking* evaluation paradigm where the Hamming distance is exhaustively computed from the query to every data-point in the dataset (Section 3.4.1). The data-points are then ranked in ascending order of Hamming distance and the resulting ranked list is used to compute ranking-based evaluation metrics.

The next point I addressed in Section 3.5 was how to split the datasets into random partitions. In a retrieval setting I need a set of held-out test queries and a database over which retrieval will be performed. The accepted methodology in the literature (the *literature standard splitting strategy*) was to randomly select a set of held-out test queries and to use the remaining data-points as the database to be ranked *and* as the training dataset for learning the hash functions (Section 3.5.1). I identified a potential overfitting concern with this strategy and advocated an approach (the *improved splitting strategy*) where a certain split of the dataset forms a held-out database that cannot be used to learn the hash functions at training time (Section 3.5.2).

The final part of this chapter, in Section 3.6, introduced the evaluation metrics I will use to quantify the retrieval effectiveness of my algorithms with respect to prior art. In this thesis I use the standard Information Retrieval (IR) metrics of *area under the precision recall curve (AUPRC)* and *mean average precision (mAP)* to evaluate the quality of the hashcodes (Sections 3.6.3-3.6.4).

3.8 Conclusion

Having defined the research landscape within which this thesis is firmly embedded in Chapters 2-3 I am now in a position to introduce my own novel contributions to the field. I begin in Chapter 4 with a new multi-threshold quantisation algorithm that relaxes the limiting assumption of Single Bit Quantisation (SBQ) (Chapter 2, Section 2.5.1) in that only one threshold should be used for binarisation per projected dimension, and furthermore that the threshold position should remain unoptimised. My model assigns more than one threshold per dimension and dynamically optimises their positions based on the distribution of the input data, showing a significant increase in retrieval effectiveness versus a host of state-of-the-art quantisation models.

Chapter 4

Learning Multiple Quantisation Thresholds

The research presented in this Chapter has been previously published in Moran et al. (2013a).

4.1 Introduction

In this chapter I make a first attempt at improving the retrieval effectiveness of the data-independent and data-dependent (unsupervised) projection models introduced in Chapter 2, Section 2.4 and Section 2.6.3 by introducing a new method for quantising their projections into binary hashcodes. I introduced the process of quantisation for hashing-based ANN search in Chapter 2, Section 2.5. In that section I discussed how it was common to quantise the projections using single bit quantisation (SBQ) in which a single threshold is placed directly at zero on a projected dimension for mean centered data. Projected values above the threshold contributed a '1' to the binary encoding for their corresponding data-point and a '0' otherwise. An argument was made that a static placement of a threshold directly at the region of highest point density is a sub-optimal approach due to the high likelihood of separating related data-points on either side of the threshold, thereby causing related data-points to be assigned different bits and ultimately negatively impacting hashing-based ANN search effectiveness.

To improve upon SBQ in this chapter I will relax the assumption of using *one statically placed threshold* for binarising a projected dimension (assumption A_1 presented in Chapter 1) by both optimising the threshold position and by exploring the benefits of allocating *one or more thresholds* per projected dimension, specifically $T = 1, 2, 3, 7$

Method	**Data-Dependent**	**Supervised**	**# Thresholds**	**Codebook**
SBQ			1	0/1
DBQ	✓		2	00/11/10
HQ	✓		3	00/01/10/11
MHQ	✓		$2^B - 1$	NBC
NPQ	✓	✓	1,2,2^{B-1}	Any

Table 4.1: Comparison of the quantisation algorithm introduced in this chapter (NPQ) versus the most closely related quantisation models from the literature. All of the baselines were previously reviewed in Chapter 2. $B \geq 2, B \in \mathbb{Z}$ denotes the number of bits per projected dimension. NBC stands for natural binary code.

and 15 thresholds[1]. As I previously discussed in Chapter 2 a quantisation scheme must provide an associated binary codebook $\mathcal{C}$, which assigns codewords to the thresholded regions of a projected dimension and a method of positioning the threshold(s). For SBQ, the codebook is simple 0/1 binary encoding $\{\mathbf{c}_i : \mathbf{c}_i \in \{0,1\}\}$ and the threshold is placed at zero, without any optimisation of the positioning. In this chapter I will explore a *multi-bit* codebook for the thresholded regions $\left\{\mathbf{c}_i : \mathbf{c}_i \in \{0,1\}^B\right\}$ where $B \geq 1$ bits (or T thresholds) are allocated per projected dimension. In the experiment evaluation I observe the corresponding change in retrieval effectiveness versus a vanilla single bit $B = 1$ per projected dimension encoding. In tandem with this I will also ascertain the benefit of *optimising* the threshold positions, rather than simply assuming that a static placement will be optimal. Table 4.1 presents a comparison of the proposed model (NPQ) to a selection of representative models from the literature.

The remainder of this Chapter is organised as follows: I begin in Section 4.2 by formulating my proposed multi-threshold quantisation algorithm. This section is broken down into Section 4.2.2 which introduces the proposed semi-supervised objective function which directly maximises the number of related data-points assigned the same bits, while minimising the occurrence of unrelated data-points being assigned the same bits. I detail how this objective function is optimised by stochastic search in Section 4.2.3. I examine the effectiveness and efficiency of the quantisation algorithm in Section 4.3 with a quantitative evaluation over the unimodal datasets presented in Chapter 3, Section 3.2. I then conclude this chapter in Section 4.4 with a discussion and con-

[1]The threshold quantities of $T = 1, 2, 3, 7, 15$ is entirely dictated by the binary codebooks used. See Chapter 2, Sections 2.5.1-2.5.4.

clusion on the main experimental findings.

4.2 Quantisation Threshold Optimisation

4.2.1 Problem Definition

My objective in this chapter is to *learn* a set of thresholds $\mathbf{t}_k = [t_{k1}, t_{k2}, \dots, t_{kT}]$ where $t_{ki} \in \mathbb{R}$ and $t_{k1} < t_{k2} \dots < t_{kT}$ for each of the K projected dimensions $\{\mathbf{y}^k \in \mathbb{R}^N\}_{k=1}^K$. The quality of the quantisation will be judged by using the resulting hashcodes to retrieve the nearest neighbours to a set of image queries. Note here we are already assuming that an existing projection function (such as LSH or ITQ) has already generated the projections, but crucially they are yet to be binarised. The learnt thresholds will be used to quantise the real-valued projections into binary using a specified codebook $\left\{\mathbf{c}_i : \mathbf{c}_i \in \{0,1\}^B\right\}$. In addition to the codebook, I formulate in this section an optimisation algorithm that will learn the quantisation thresholds so that neighbouring points $\mathbf{x}_i \in \mathbb{R}^D, \mathbf{x}_j \in \mathbb{R}^D$ are more likely to have similar hashcodes $\mathbf{b}_i \in \{0,1\}^K, \mathbf{b}_j \in \{0,1\}^K$. This optimisation problem is challenging due to the prohibitively large search space $O(N^T)$ of possible thresholds and the non-differentiable nature of my desired semi-supervised objective function. In Sections 4.2.2-4.2.3 I discuss the intractability of the problem and introduce an algorithmic solution that is both readily scalable and demonstrably effective.

4.2.2 Judging Threshold Quality: F_1-Measure Objective Function

In contrast to previous quantisation models such as AGH (Liu et al. (2011)), DBQ (Kong et al. (2012)) and MHQ (Kong and Li (2012a)), my quantisation algorithm, which I will refer to as *Neighbourhood Preserving Quantisation (NPQ)*, leverages a binary adjacency matrix $S \in \{0,1\}^{N_{trd} \times N_{trd}}$, where N_{trd} is the number of training data-points ($N_{trd} \ll N$), to guide the threshold positioning. My hypothesis is that the neighbourhood structure between the data-points in the input feature space is a valuable signal for guiding the quantisation thresholds within the lower-dimensional projected space. The adjacency matrix $\mathbf{S}$ therefore encodes the neighbourhood structure of the data-points in the original feature space, where $S_{ij} = 1$ if points $\mathbf{x}_i$ and $\mathbf{x}_j$ are considered neighbours (a *positive* pair), and $S_{ij} = 0$ otherwise (a *negative* pair)[2]. $\mathbf{S}$ can be

[2] I set diagonal matrix elements to zero ($S_{ii} = 0$) for all computations in this chapter.

generated, for example, by computing Euclidean distance between N_{trd} data-points and setting any data-points within an ε-ball of each other as true nearest neighbours[3]. The pairwise affinity matrix $\mathbf{S}$ specifies the pairs of points that should fall within the same thresholded regions and therefore be assigned identical hashcodes from the codebook.

We can now define the desired objective function for threshold positioning that directly leverages the neighbourhood structure encoded in $\mathbf{S}$. For a fixed set of thresholds $\mathbf{t}_k = [t_{k1} \dots t_{kT}]$ I define a per-projected dimension indicator matrix $\mathbf{P}^k \in \{0,1\}^{N_{trd} \times N_{trd}}$ with the property given in Equation 4.1:

$$P^k_{ij} = \begin{cases} 1, & \text{if} \quad \exists \gamma \quad s.t. \quad t_{k\gamma} \leq (y^k_i, y^k_j) < t_{k(\gamma+1)} \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

The index $\gamma \in \mathbb{Z}$ spans the range: $0 \leq \gamma \leq T$, where the scalar quantity T denotes the total number of thresholds partitioning a given projected dimension. Intuitively, matrix $\mathbf{P}^k$ indicates whether or not the projections (y^k_i, y^k_j) of any pair of data-points $(\mathbf{x}_i, \mathbf{x}_j)$ fall within the same thresholded region of the one-dimensional projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$. Given a particular instantiation of the thresholds $[t_{k1} \dots t_{kT}]$, the algorithm counts the number of *true positives* (TP), *false negatives* (FN) and *false positives* (FP) across all regions. The requisite TP, FP and FN counts can then be stated as in Equations 4.2-4.4

$$TP = \frac{1}{2}\sum_{ij} P_{ij} S_{ij} = \frac{1}{2}\|\mathbf{P} \circ \mathbf{S}\|_1 \tag{4.2}$$

$$FN = \frac{1}{2}\sum_{ij} S_{ij} - TP = \frac{1}{2}\|\mathbf{S}\|_1 - TP \tag{4.3}$$

$$FP = \frac{1}{2}\sum_{ij} P_{ij} - TP = \frac{1}{2}\|\mathbf{P}\|_1 - TP \tag{4.4}$$

where $\circ$ denotes the Hadamard (elementwise) product and $\|.\|_1$ is the L_1 matrix norm defined as $\|\mathbf{X}\|_1 = \sum_{ij} |X_{ij}|$. Intuitively TP is the number of positive pairs that are found within the same thresholded region, FP is the proportion of negative pairs found within the same region, and FN are the proportions of positive pairs found in different regions. The factor of 1/2 appears in Equations 4.2-4.4 as both $\mathbf{P}$ and $\mathbf{S}$ are symmetric matrices under the ε-*NN* groundtruth paradigm and so each pairwise relationship between two

[3]A fuller definition of ε-NNs can be found in Chapter 3, Section 3.3.1

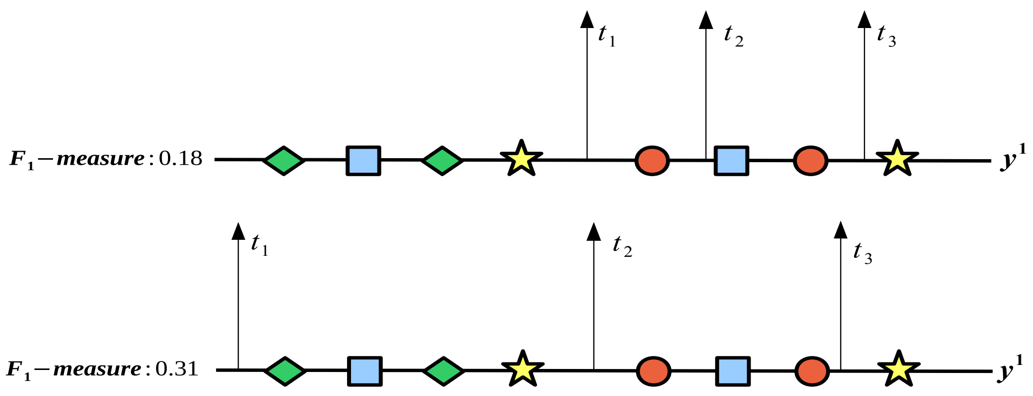


Figure 4.1: Maximisation of the F_1-measure can lead to an effective setting of the quantisation thresholds. In both diagrams we seek to position three thresholds along the same projected dimension. In the top diagram the threshold positioning leads to an F_1-measure of 0.18. This is a rather low score which results from separating many of the true NNs (indicated with the same colour and shape) in different regions. The threshold positions in the lower diagram lead to a higher F_1-measure, approximately twice as high, which arises from capturing more true nearest neighbours in the same thresholded regions.

points is counted twice: for example if $\mathbf{x}_i$ and $\mathbf{x}_j$ are true nearest neighbours then $S_{ij} = 1$ and $S_{ji} = 1$. The TP, FP and FN counts are combined using the familiar set-based F_1-measure[4] from Information Retrieval (Equation 4.5):

$$F_1(\mathbf{t}_k) = \frac{2\|\mathbf{P} \circ \mathbf{S}\|_1}{\|\mathbf{S}\|_1 + \|\mathbf{P}\|_1} \tag{4.5}$$

The application of an F_1-measure[5] based objective function is motivated by the highly unbalanced nature of the adjacency matrix $\mathbf{S}$: this matrix is usually very sparse, with approximately 1% of the elements being positive pairs. The F_1-measure is well known to be much less affected by this imbalanced distribution between positive and negatives (as we are not affected by true negatives) than, for instance, the classification accuracy (Chawla (2005)). I present a simple example in Figure 4.1 that illustrates the computation of the F_1-measure on a toy projected dimension. The overall objective function that I seek to optimise is given in Equation 4.6.

[4]I use F_β-measure with $\beta = 1.0$ throughout this Chapter. In Chapter 5, I explore the extent to which retrieval performance can be increased by tuning this parameter based on the data distribution.

[5]Specifically I use micro F_1-measure which collates the TPs, FPs and FNs across data-points before computing the precision and recall. The benefits of computing a macro F_1-measure in the context of multiple threshold learning is left for future work.

$$\mathcal{J}_{npq}(\mathbf{t}_k) = \alpha F_1(\mathbf{t}_k) + (1-\alpha)(1-\Omega(\mathbf{t}_k)) \tag{4.6}$$

where $\alpha \in [0,1]$ and the unsupervised term $\Omega(\mathbf{t}_k)$ is defined as given in Equation 4.7:

$$\Omega(\mathbf{t}_k) = \frac{1}{\sigma_k} \sum_{j=1}^{T+1} \sum_{i:y_i^k \in \mathbf{r}_j} \left\{y_i^k - \mu_j\right\}^2 \tag{4.7}$$

where $\mathbf{r}_j = \left\{y_i | t_{j-1} \leq y_i < t_j, y_i \in \mathbf{y}^k\right\}$ denotes the projections within thresholded region $\mathbf{r}_j$ with $t_0 = -\infty, t_{T+1} = +\infty$, $\sigma_k = \sum_{i=1}^{N_{trd}} \left\{y_i^k - \mu_k\right\}^2$, $\mu_k \in \mathbb{R}$ denotes the mean of projected dimension $\mathbf{y}^k \in \mathbb{R}^N$ and $\mu_j \in \mathbb{R}$ denotes the mean of the projections located in thresholded region $\mathbf{r}_j$. Intuitively maximisation of Equation 4.6 encourages a clustering of the projected dimension so that as many of the must-link (i.e. $S_{ij} = 1$) and cannot-link (i.e. $S_{ij} = 0$) constraints encoded in the adjacency matrix $\mathbf{S}$ are respected while also minimising the cluster dispersion of the projections within each thresholded region. Equation 4.6 therefore fuses two valuable signals in a complementary manner: the neighbourhood structure encoded in the adjacency matrix which provides information on the pairwise relationships between the data-points in the input feature space; and the neighbourhood information captured by the projection function that was responsible for generating the projected dimensions in the first place. In this way we avoid relying entirely on the ability of the projection function to correctly place nearby data-points within close proximity of each other along a projected dimension. This semi-supervised objective function is the main point of conceptual departure from existing quantisation algorithms such as AGH (Chapter 2, Section 2.5.2), MHQ (Chapter 2, Section 2.5.4) and DBQ (Chapter 2, Section 2.5.3) which only leverage the structure in the projected space. I investigate the synergy between these two signals and their resulting effect on retrieval performance in my experimental evaluation (Section 4.3).

The F_1-measure term in Equation 4.6 is non-differentiable due to the discontinuous form of Equation 4.1 at the threshold points $t_{k\gamma}, t_{k(\gamma+1)}$. Continuous optimisation via gradient ascent is therefore difficult. I will demonstrate in Section 4.2.3 that we can directly optimise this objective function without appealing to a continuous relaxation.

4.2.3 Efficient Threshold Optimisation through Stochastic Search

For a given projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$, there is an optimal setting of the thresholds $\mathbf{t}_k^* = [t_{k1} \dots t_{kT}]$ that will maximise Equation 4.6. There are two issues we need to tackle to optimise this function, Firstly, as I discussed, my desired objective function is non-differentiable making a gradient descent approach infeasible. Secondly, brute force maximisation is of $O(N_{trd}^2 N_{trd}^T T)$[6] time complexity ($T \in [1,2,\ldots,15]$), which due to the high degree polynomial does not scale up to large training datasets and multiple quantisation thresholds per dimension. I tackle both issues by exploring two non-deterministic optimisation frameworks, namely *simulated annealing* (Kirkpatrick et al. (1983)) and *evolutionary algorithms (EA)* (Goldberg (1989)). Both stochastic search methods are well known techniques for discovering approximate solutions to challenging combinatorial optimisation problems. Neither stochastic search framework requires the function to be continuous or have a derivative and has parameters that can trade-off computation time versus accuracy achieved. If I denote by F a parameter that controls the number of evaluations of Equation 4.6 within the optimisation framework, we are able to achieve a more reasonable time complexity of $O(N_{trd}^2 TF)$ for learning the optimal threshold positions for a single projected dimension[7]. Remarkably, as we will see in the experimental evaluation, despite the approximate nature of the stochastic search algorithm we are able to find a good local optimum within an acceptable number of objective function evaluations (F). In total, for $K' = \lfloor K/B \rfloor$ projected dimensions, the time complexity is of $O(K' N_{trd}^2 TF)$, where B denotes the number of bits per projected dimension

I will now describe the specifics of how I use simulated annealing and evolutionary algorithms to optimise Equation 4.6. The stochastic search is described in the context of a single (arbitrary) projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$ since each projected dimension is quantised independently. To learn a set of thresholds for a given projected dimension, the stochastic search algorithm initially generates H candidate *sets* of thresholds uniformly at random in the matrix $\mathbf{T}^k \in \mathbb{R}^{H \times T}$ where $\mathbf{T}^k_{r\bullet} = [t_{r1} \ldots t_{rT}]$ with $r \in [1 \ldots H]$. The number H of candidate threshold sets (i.e. rows in matrix $\mathbf{T}^k$) is 1 for simulated annealing and $H \geq 1$ for evolutionary algorithms. Each row of the threshold matrix $\mathbf{T}^k \in \mathbb{R}^{H \times T}$ represents a starting point in the T dimensional threshold space. The objective of the stochastic search is to navigate through this space of thresholds to points representing local maxima in the objective function (Equation 4.6). The threshold configuration in the row of $\mathbf{T}^k$ that yields the greatest local maximum as judged by Equation 4.6 is selected as the quantisation for projected dimension $\mathbf{y}^k$. At each iteration the stochastic search algorithm evaluates each row of thresholds $\mathbf{T}^k_{r\bullet}$ by constructing

[6]Typically the adjacency matrix $\mathbf{S}$ is highly sparse and so the number of non-zero elements are much less than N_{trd}^2.

[7]The unsupervised part of the objective function is linear $O(FN_{trd})$ and so I ignore it in my statement of the overall time complexity.

the matrix $\mathbf{P}^{rk}$ and computing the corresponding objective function value (Equation 4.6). Each threshold in the matrix $\mathbf{T}^k$ is then subsequently perturbed to shift the search into a new region of threshold space. The manner in which each row of thresholds in $\mathbf{T}^k$ are modified to move into a position in threshold space possibly exhibiting a higher objective function value is particular to the stochastic search algorithm. In the next two sections I will briefly describe how I adapt simulated annealing (Section 4.2.3.1) and evolutionary algorithms (Section 4.2.3.2) for my task. Application of both stochastic search methods for threshold finding in the context of hashing-based ANN search is novel to my knowledge.

4.2.3.1 Simulated Annealing

Simulated annealing is a popular non-deterministic optimisation algorithm used to find a good, but not necessary global optimum for problems that exhibit a large search space which would otherwise take an inordinate amount of computation to traverse exhaustively (Ingber (1993), Kirkpatrick et al. (1983)). This method of stochastic search has been used successfully in many diverse applications from finding the optimal wiring of a computer chip (Kirkpatrick et al. (1983)) to finding the conformational substates of proteins (Bohr and Brunak (1989)). Simulated annealing is named after the process of annealing in Metallurgy whereby a crystalline solid is gradually cooled to form a low energy highly structured crystal lattice with minimal defects. The maximum temperature and the cooling schedule are critical parameters of the physical annealing process if the ground energy state is to be achieved. The computational version of simulated annealing exhibits *three* important factors that affect the stochastic search: the maximum temperature, the scheme for reducing the temperature and the scheme for proposing updates (perturbing the current solutions). There have been many proposals in the literature for reducing the temperature and exploring the solution space (Ingber (1993)). In this thesis I select the perturbation function that modifies the thresholds $\mathbf{t}_k$ (i.e. selects the "neighbours" of the current state) by a magnitude given by the current temperature $S \in \mathbb{R}$, with a direction that is chosen uniformly at random. The perturbed set of thresholds $\mathbf{t}_k^{'}$ is accepted either if the new objective function value $\mathcal{J}(\mathbf{t}_k^{'})$ is greater than the previous value $\mathcal{J}(\mathbf{t}_k)$, or at random with a probability that is dependent on the current temperature and the difference between the old and new objective function values for the thresholds. The probability of a sub-optimal solution being accepted is given in Equation 4.8.

Algorithm 6: MULTIPLE THRESHOLD LEARNING VIA SIMULATED ANNEALING

```
Input: Projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$, initial temperature $S_0 \in \mathbb{R}$, number of
       iterations $M \in \mathbb{Z}_+$
Output: Optimised quantisation thresholds $\mathbf{t}_k \in \mathbb{R}^T$ for projected dimension
        $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$
1  $S = S_0$
2  Initialise thresholds $\mathbf{t}_k \in \mathbb{R}^T$ uniformly at random
3  for $m \leftarrow 1$ to $M$ do
4      $\{\Delta : \Delta_j \sim Unif(0,1), j \in [1,T]\}$        // Draw perturbation vector
5      $\Delta = \Delta / ||\Delta||_2$
6      $\mathbf{t}_k^{'} = \mathbf{t}_k + S\Delta$                // Generate solution candidate
7      $r \sim Unif(0,1)$
8      $\Delta_k = \mathcal{J}(\mathbf{t}_k^{'}) - \mathcal{J}(\mathbf{t}_k)$
9      if $(\mathcal{J}(\mathbf{t}_k^{'}) > \mathcal{J}(\mathbf{t}_k))$ then
10         $\mathbf{t}_k = \mathbf{t}_k^{'}$
11     else if $(r < (1/(1+exp(\frac{\Delta_k}{S}))))$ then
12         $\mathbf{t}_k = \mathbf{t}_k^{'}$        // Accept solution based on Boltzmann density
13     end
14     $S = S_0 \times 0.95^M$                   // Anneal temperature
15 end
16 return $\mathbf{t}_k$
```

$$\Pr(\Delta_k, S) = \frac{1}{(1+exp(\frac{\Delta_k}{S}))} \tag{4.8}$$

where $\Delta_k = \mathcal{J}(\mathbf{t}_k^{'}) - \mathcal{J}(\mathbf{t}_k)$. The probability distribution in Equation 4.8 is known as *Boltzmann annealing* (Szu and Hartley (1987)). While moving the search to a region of lower objective function value may seem counterintuitive, it is exactly this possibility that permits the search to escape local maxima in the hope of discovering a greater local maximum. The temperature is lowered (cooled) by the function: $S = S_0 \times 0.95^m$ where S_0 is the initial temperature and S is the current temperature at iteration $m \in \{1, \ldots, M\}$. At high temperature the stochastic search will explore more of the parameter space, and as the temperature is gradually lowered the exploration will become more and more restricted with a lower probability of jumping to sub-optimal regions of threshold space. The search terminates when there is no significant difference between the objective function values or a maximum number of iterations has been exceeded. The threshold configuration at termination of the search is used to quantise the projected dimension $\mathbf{y}^k$. Simulated annealing requires no derivative information on the function to be evaluated, making it an ideal candidate for directly maximising Equation 4.6.

My adaptation of simulated annealing for threshold finding is presented in Algorithm 6. In Line 4, the perturbation vector Δ is drawn uniformly at random and specifies how the existing set of thresholds are to be adjusted. In Line 6 the perturbation vector is multiplied by the temperature S, which dictates the magnitude of the threshold change. Finally in Lines 9-13 the new set of thresholds ($\mathbf{t}_k^{'}$) are either excepted if the objective function value ($\mathcal{J}(\mathbf{t}_k^{'})$) is greater than what it was previously ($\mathcal{J}(\mathbf{t}_k)$), or if not, with a probability based on Equation 4.8. I use the Matlab simulated annealing toolkit for all the simulated annealing experiments in this dissertation[8].

4.2.3.2 Evolutionary Algorithms

Evolutionary algorithms employ a method reminiscent to "natural selection" and the Darwinian principle of the survival of the fittest to generate gradually better solutions ("individuals") to a combinatorial search problem. The intuition behind this method of stochastic search is to increase the average fitness of a set of individuals by repeatedly breeding together individuals using operators inspired by natural genetics, such as *crossover* and *mutation*. The fitness of the individuals is judged using the application-specific objective function, which is Equation 4.6 for the purposes of this chapter. This iterative breeding process has the net effect that over a number of iterations M the population of individuals will have a higher average fitness than their parents from earlier generations ("iterations"). In my application the individuals are sets of thresholds $\mathbf{T}^k_{r\bullet}$, each of which represents a particular quantisation of the projected dimension $\mathbf{y}^k$. The goal is to find a set of thresholds that give the highest value for the objective function. More than one individual is generated ($H > 1$) by instantiating the matrix $\mathbf{T}^k \in \mathbb{R}^{H\times T}$ with entries selected uniformly at random. Each individual represents a particular point in threshold space, and therefore the evolutionary algorithm maintains multiple parallel hypotheses as to the optimal quantisation of the projected dimension. The H

[8]http://uk.mathworks.com/discovery/simulated-annealing.html

individuals in the current population represented by $\mathbf{T}^k \in \mathbb{R}^{H \times T}$ are used to generate new, potentially higher quality individuals by iteratively repeating the following four steps:

1. **Sampling**: Probabilistically select for reproduction a predefined proportion $H^{'} = max(\lfloor \omega H + 0.5 \rfloor, 2)$ of the H sets of thresholds in $\mathbf{T}^k \in \mathbb{R}^{H \times T}$ with a probability dependent on their relative objective function values. The parameter $\omega \in \mathbb{R}$ and is set to the default of 0.9 in this thesis. The greater the fitness value of an individual the higher the likelihood that it will be selected as a hypothesis for the optimal configuration of quantisation thresholds. I opt for the standard *stochastic universal sampling* (SUS) method in this thesis. Briefly, SUS is a form of roulette wheel selection in which every individual is allocated a portion of the hypothetical wheel in proportion to its computed fitness value. Individuals with a higher fitness are allocated a correspondingly larger portion of the wheel. In this fitness proportionate form of sampling, the wheel is spun only once and individuals located at equally spaced intervals (computed based on the number of desired individuals to be sampled) around the wheel, from the point at which the wheel stopped, are selected for breeding. Individuals may be selected multiple times particularly if they have a high fitness values. SUS guarantees that the observed selection frequencies of individuals accord with the expected selected frequencies: so for example if an individual occupies 20% of the wheel and we wish to sample 100 individuals then that individual will be selected 20 times on average. This guarantee is not given for the vanilla roulette wheel selection algorithm in which the wheel is spun as many times as the number of individuals we wish to sample.

2. **Crossover**: The $H^{'}$ individuals selected in the previous sampling step are placed into $\lfloor H^{'}/2 \rfloor$ pairs and the *single point crossover* operator is then applied to each pair with probability $\theta \in [0,1]$[9]. Given two sets of thresholds $\mathbf{t}_k = [t_1^k \ldots t_T^k]$, $\mathbf{t}_k^{'} = [t_1^{k'} \ldots t_T^{k'}]$, single point crossover picks an integer index $i \in [1,T]$ uniformly at random and forms two new pairs (the "offspring") by swapping elements using the index i as the crossover point giving two new sets of thresholds: $\mathbf{t}_k = [t_1^k \ldots t_i^k, t_{i+1}^{k'} \ldots t_T^{k'}]$ and $\mathbf{t}_k^{'} = [t_1^{k'} \ldots t_i^{k'}, t_{i+1}^k \ldots t_T^k]$. In practice the pairs are formed by taking individuals in the even numbered rows and crossing them with the individuals in the adjacent odd number rows: so for example the thresholds in row

[9]Crossover is usually applied with a high probability $\theta \approx 0.7$ (Freitas (2002)).

$\mathbf{T}^k_{1\bullet}$ are crossed with those in row $\mathbf{T}^k_{2\bullet}$, and the same for those in rows $\mathbf{T}^k_{3\bullet}$, $\mathbf{T}^k_{4\bullet}$ and so forth.

3. **Mutation**: Apply the *mutation* operator with probability $\phi \in [0,1]$ to the ωH offspring produced by the crossover operator[10]. Mutation randomly changes the value of a single threshold for a particular individual. In the evolutionary algorithms literature mutation acts as a *background operator* that ensures the probability of exploring a particular subspace of the threshold space is always greater than zero. Mutation is an *exploration* operator which drives the search to previously untouched areas of the space. This is to be contrasted with the other operators which geared towards the *exploitation* of promising regions of the solution space.

4. **Reinsertion**: *Reinsert* the offspring into the current population $\mathbf{T}^k \in \mathbb{R}^{H \times T}$. If the number of offspring $H^{'}$ is less than the number of individuals (H) in the original population, i.e. there is a *generation gap* (G), then a suitable replacement strategy is evoked. In this thesis I use an *elitist* replacement strategy in which the $H^{'}$ generated offspring replaces the individuals in the population that have the lowest fitness values. The remaining $H - H^{'}$ individuals in the original population with the highest fitness are therefore deterministically propagated to the next generation.

The above four steps are repeated for a predefined number of generations M. Given the bias towards maintaining and "breeding" those individuals that have a higher fitness, we expect that over a sufficient number of generations the average fitness value of the population of individuals will increase and therefore gradually move towards regions of threshold space with high objective function values. In my case I hope that this region contains a configuration of the quantisation thresholds that assign similar data-points similar bits, and dissimilar data-points different bits. The search terminates when the maximum number of generations have been exceeded or there is no appreciable increase in the objective function value. At termination of the search the set of thresholds $\mathbf{T}^k_{r\bullet}$ with the maximum objective function value at generation M is used to quantise the corresponding projected dimension $\mathbf{y}^k$.

Algorithm 7 summarises the main steps in using evolutionary algorithms to learn multiple quantisation thresholds. Line 1 initialises a matrix $\mathbf{T}^k$ of thresholds, one set

[10]In practice mutation is typically applied with a very low probability $0.001 \leq \phi \leq 0.01$ (Freitas (2002)).

Algorithm 7: Multiple Threshold Learning via Evolutionary Algorithms

```
Input: Projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$, # iterations $M \in \mathbb{Z}_+$, number of
       threshold sets $H \in \mathbb{Z}_+$, number of thresholds per dimension $U \in \mathbb{Z}_+$,
       Mutation probability $\phi \in [0,1]$, Crossover probability $\theta \in [0,1]$,
       Proportion to select $\omega \in [0,1]$
Output: Optimised quantisation thresholds $\mathbf{t}_k \in \mathbb{R}^T$ for $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$
1 Initialise $H$ sets of thresholds $\mathbf{T}^k \in \mathbb{R}^{H \times T}$ uniformly at random
2 for $m \leftarrow 1$ to $M$ do
3     Compute $\mathbf{f} \in \mathbb{R}^H$ such that $f_j = \mathcal{J}_{npq}(\mathbf{T}^k_{j\bullet})$
4     Select $H^{'} = max(\lfloor \omega H + 0.5 \rfloor, 2)$ rows from $\mathbf{T}^k$ based on $\mathbf{f}$, place in $\mathbf{T}^{k'}$
5     Form $\lfloor H^{'}/2 \rfloor$ pairs from $\mathbf{T}^{k'}$, crossover pairs with probability $\theta$
6     Mutate thresholds in $\mathbf{T}^{k'}$ with probability $\phi$
7     Reinsert $\mathbf{T}^{k'}$ in $\mathbf{T}^k$ with elitist replacement
8 end
9 return $\mathbf{t}_k = \underset{\mathbf{t}_j}{\text{argmax}}\ \mathcal{J}_{npq}(\mathbf{T}^k_{j\bullet})$
```

of T thresholds per row. In Line 3, the objective function (Equation 4.6) is computed for each of the H sets of thresholds on the rows of matrix $\mathbf{T}^k$. In Line 4, the sampling step is performed and selects $H^{'} = max(\lfloor \omega H + 0.5 \rfloor, 2)$ rows from matrix $\mathbf{T}^k$. In Line 5, $\lfloor H^{'}/2 \rfloor$ pairs are formed from the selected threshold sets and the crossover operator is applied. The resulting $H^{'}$ sets of thresholds are mutated (Line 6), and then reinserted back into matrix $\mathbf{T}^k$ (Line 7). In Line 9 the row of $\mathbf{T}^k$ that yields the highest objective function value is selected as the set of thresholds ($\mathbf{t}_k$) used for quantising the given projected dimension. The evolutionary algorithm solver I use in this thesis is the open-source Sheffield Genetic Algorithms Toolbox[11]. I compare and contrast the efficiency and empirical performance of both simulated annealing and evolutionary algorithms in my experimental evaluation (Section 4.3).

[11] `http://codem.group.shef.ac.uk/index.php/ga-toolbox`

4.3 Experimental Evaluation

4.3.1 Experimental Configuration

In this section I perform a set of experiments to examine the effectiveness and efficiency of the multi-threshold quantisation algorithm described in Section 4.2. I directly compare my model to state-of-the-art quantisation algorithms from the literature: Single Bit Quantisation (SBQ), Hierarchical Quantisation (HQ), Double Bit Quantisation (DBQ) and Manhattan Hashing Quantisation (MHQ). All of these baselines quantisation algorithms were reviewed in detail in Chapter 2, Section 2.5. The experimental evaluation is structured to provide an answer to the following four main hypotheses:

- *H_1: A* single *threshold optimised using Equation 4.6 yields a higher retrieval effectiveness than Single Bit Quantisation (SBQ) for LSH and PCA projections.*
- *H_2:* Two *thresholds optimised using Equation 4.6 leads to a higher retrieval effectiveness than Double Bit Quantisation (DBQ) for LSH and PCA projections.*
- *H_3:* Multiple *(3, 7, 15) thresholds optimised with Equation 4.6 outperform multiple thresholds learning using the Manhattan Quantisation (MHQ) algorithm of Kong et al. (2012) for LSH and PCA projections.*
- *H_4:* Three *thresholds optimised with Equation 4.6 yield a higher retrieval effectiveness for PCA, LSH, ITQ, SH, SKLSH projections than the MHQ quantisation algorithm.*

In all four cases I use the appropriate quantisation codebook, namely the traditional binary 0/1 codebook (Indyk and Motwani (1998)) for H_1, the double bit quantisation codebook (Kong and Li (2012a)) for H_2 and the Manhattan quantisation codebook (Kong et al. (2012)) for H_3, H_4[12]. My quantisation model is general and can potentially be used with *any* binary codebook, *any* number of thresholds and indeed *any* projection function of interest. To this end these four hypotheses will examine different configurations of my quantisation algorithm in which the codebook (binary, DBQ and MHQ) and number of thresholds ($T = 1, 2, 3, 7$ and 15 thresholds) are varied. In doing so I hope to understand the exact circumstances in which the proposed quantisation algorithm is most effective while also discovering where it is most likely to

[12] I use the Manhattan quantisation codebook for H_4 because it constitutes the best prior art for multi-threshold quantisation.

Parameter	**Setting**	**Chapter Reference**
Groundtruth Definition	ϵ-NN	Chapter 3, Section 3.3
Evaluation Metric	AUPRC	Chapter 3, Section 3.6.3
Evaluation Paradigm	Hamming Ranking	Chapter 3, Section 3.4.1
Random Partitions	10	Chapter 3, Section 3.5
Number of Bits (K)	16-128	Chapter 2, Section 2.4

Table 4.2: Configuration of the main experimental parameters for the results presented in this section.

fail. In addition to these four hypotheses I will also measure the impact of the main parameters of my method, specifically the effect of the training database size N_{trd} and the influence of the interpolation parameter α. I will also be interested in the *training time* of the threshold optimisation algorithm, given the importance of efficiency for any method of hashing-based ANN search.

To constrain the quantity of experiments I closely follow the experimental protocol in the relevant literature (Kong et al. (2012); Kong and Li (2012a,b); Kulis and Darrell (2009); Raginsky and Lazebnik (2009); Gong and Lazebnik (2011)) and make a number of choices with regards to the groundtruth and evaluation paradigm. Unless otherwise stated in the relevant experiment I use the experimental framework detailed in Table 4.2 for evaluation. Specifically, the experiments will be conducted on the three unimodal image datasets (CIFAR-10, NUS-WIDE and SIFT1M) described in Chapter 3, Section 3.2.1 using the ϵ-NN groundtruth definition presented in Chapter 3, Section 3.3. The Hamming ranking evaluation paradigm (Chapter 3, Section 3.4.1) and area under the precision recall curve (AUPRC) (Chapter 3, Section 3.6.3) will be used to ascertain the quality of the hashcodes. In all experiments I also follow previously accepted procedure (Kong et al. (2012), Kong and Li (2012a), Kong and Li (2012b)) and randomly select $N_{teq} = 1,000$ data points as testing queries ($\mathbf{X}_{teq} \in \mathbb{R}^{N_{teq} \times D}$), with the remaining points ($\mathbf{X}_{db} \in \mathbb{R}^{N_{db} \times D}$) being used as the database upon which to learn and test the hash functions according to the selected dataset splitting strategy (namely the literature standard or improved splitting strategy). A further breakdown of the specific dataset splits I use is shown in Tables 4.3-4.4.

Partition	**CIFAR-10**	**NUS-WIDE**	**SIFT1M**
Test queries (N_{teq})	1,000	1,000	1,000
Validation queries (N_{vaq})	1,000	1,000	1,000
Validation database (N_{vad})	10,000	10,000	10,000
Training database (N_{trd})	2,000	10,000	10,000
Test database (N_{ted})	46,000	247,648	978,000

Table 4.3: 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. There is no overlap between the data-points across partitions.

Partition	**CIFAR-10**	**NUS-WIDE**	**SIFT1M**
Test queries (N_{teq})	1,000	1,000	1,000
Validation queries (N_{vaq})	1,000	1,000	1,000
Validation database (N_{vad})	10,000	10,000	10,000
Training database (N_{trd})	2,000	10,000	10,000
Test database (N_{ted})	59,000	268,648	999,000

Table 4.4: 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.

4.3.2 Parameter Optimisation

The quantisation thresholds are then learnt on the training dataset ($\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D}$). The thresholds are then subsequently used to quantise the test dataset projections ($\mathbf{X}_{teq} \in \mathbb{R}^{N_{teq} \times D}, \mathbf{X}_{ted} \in \mathbb{R}^{N_{ted} \times D}$). All reported AUPRC figures are computed using repeated random sub-sampling cross-validation averaged over ten independent runs. To determine the statistical significance of my results I use a Wilcoxon signed rank test (Smucker et al. (2007)). When comparing system A to system B on a given random split of the dataset, the unit of the significance test is a pair of AUPRC values, one from a retrieval run by System A and the other from a retrieval run by System B. In all presented result tables the symbol ▲▲/▼▼ indicates a statistically significant increase/decrease with $p < 0.01$, while ▲/▼ indicates a statistically significant increase/decrease with $p < 0.05$. Further hypothesis specific experimental settings, for example the setting of the interpolation parameter α, will be detailed in the relevant section.

4.3.3 Experimental Results

4.3.3.1 Effect of the Amount of Supervision (N_{trd})

In this experiment I will examine the effect of the amount of supervisory information N_{trd} on the retrieval effectiveness of my semi-supervised threshold optimisation algorithm. Recall from Section 4.2 that the adjacency matrix $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ specifies which pairs of data-points $\mathbf{x}_i \in \mathbb{R}^D, \mathbf{x}_j \in \mathbb{R}^D$ should be assigned the same binary codes ($S_{ij} = 1$) and which pairs should have different binary codes $S_{ij} = 0$. My chosen objective function (Equation 4.6) uses this information to position the quantisation thresholds along a projected dimension to maximise the number of true pairs ($S_{ij} = 1$) that fall within the same quantised regions (and therefore assigned the same bits) while minimising the number of unrelated data-points ($S_{ij} = 0$) falling within the same thresholded regions. The experimental configuration used in this section is presented in Table 4.5. I use the simplest possible parametrisation of the model. Concretely, I configure the model to optimise the position *one* threshold per projected dimension and use the standard binary 0/1 codebook with Hamming distance for pairwise comparisons. To isolate the effect of N_{trd} I set the interpolation parameter in Equation 4.6 to $\alpha = 1$ throughout this experiment. The threshold configuration maximising Equation 4.6 is obtained using evolutionary algorithms (Section 4.2.3.2) with setting of $H = 15$ (number of populations) and $M = 15$ (number of generations). I study the effect of the stochastic search method in Section 4.3.3.3 and the effect of the α parameter in Section 4.3.3.2.

Method	**# Thresholds/dim**	**Encoding**	**Ranking Strategy**
NPQ	1	0/1	Hamming

Table 4.5: Parametrisation of the quantisation methods studied in Section 4.3.3.1. NPQ stands for Neighbourhood Preserving Quantisation and is the novel algorithm proposed in this chapter.

The *validation* dataset AUPRC obtained with various levels of supervision is shown in Figure 4.2 for all three image datasets. The trend exhibited by the graph accords with expectations in that there is a steady increase in AUPRC as more supervision is used for the threshold learning. In all three cases the AUPRC starts to level between

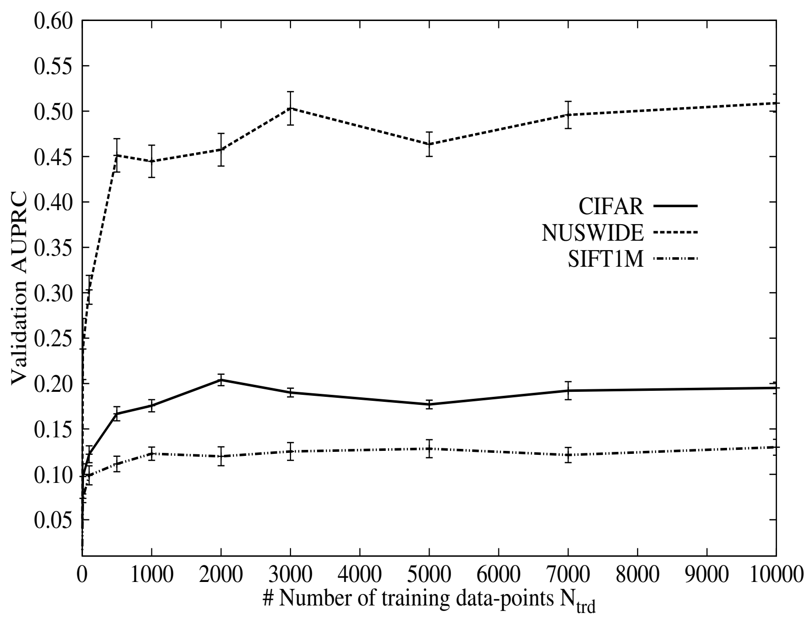


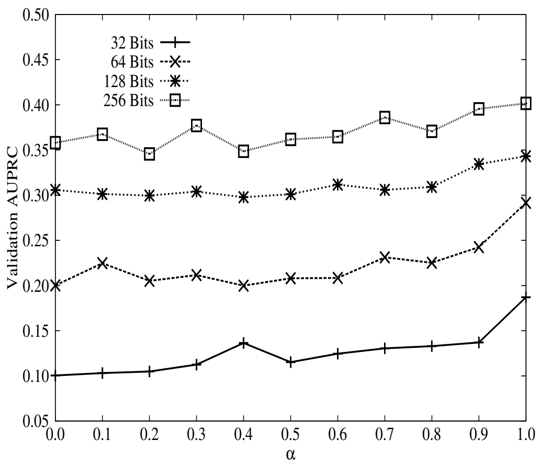
Figure 4.2: The effect of the amount of supervision N_{trd} on the *validation* dataset retrieval effectiveness (AUPRC). The results are shown for LSH projections with $T = 1$ threshold per projected dimension (hashcode length of 32 bits). The bars show the standard error of the mean.

$N_{trd} = 2,000$-$10,000$ data-points suggesting there are limited gains in retrieval effectiveness to be had with increasing levels of supervision after that point. This result is encouraging from an efficiency standpoint given that the larger the adjacency matrix the greater the amount of computation and memory required to learn the quantisation thresholds[13]. For all three datasets this experiment suggests that a relatively small adjacency matrix of around 1-2% of the total dataset size is sufficient for learning effective quantisation thresholds. In the remaining experiments in this chapter I set $N_{trd} = 2000$ for CIFAR-10 and $N_{trd} = 10,000$ for the larger NUS-WIDE and SIFT1M datasets, as this is the amount of training data at which the maximum validation dataset AUPRC is reached in each case. The training time of the multi-threshold quantisation algorithm with these settings of N_{trd} is examined in Section 4.3.3.4.

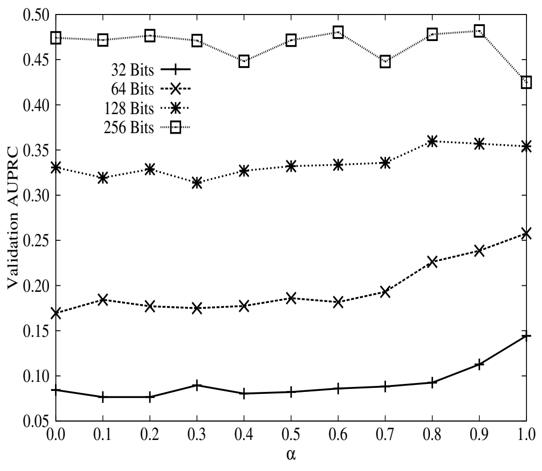
4.3.3.2 Effect of the α Interpolation Parameter

In this section I study the effect of varying the interpolation parameter $\alpha \in [0, 1]$ in Equation 4.6 for LSH projections. This parameter interpolates between the supervised

[13]The time complexity of threshold learning is $O(N_{trd}^2 TF)$, where $F = HM$ is the number of objective function evaluations made by the Evolutionary Algorithm. Memory requirements scale as $O(N_{trd}^2)$. Typically the adjacency matrix $\mathbf{S}$ is highly sparse and so the number of non-zero elements $S \ll N_{trd}^2$.



(a) **Binary (0/1) codebook**



(b) **DBQ (00/11/10) codebook**

Figure 4.3: Variation in AUPRC with the value of α for the CIFAR-10 dataset (LSH projections) and the binary (0/1) codebook (Figure (a)) and the DBQ (00/11/10) codebook (Figure (b)) .

F_1-measure term obtained from counting the number of true positives, false positives and false negatives within each thresholded region, with the unsupervised normalised variance term obtained from the projections of the data-points. Studying the preferred setting of this parameter will shed light on which signal (unsupervised or supervised) is the most important for effective placement of the quantisation thresholds, or whether a convex combination of the two signals is best. Intuitively one might expect the majority of the weight to be assigned to the more reliable supervised signal. The optimum threshold configuration is obtained using evolutionary algorithms (EA), with the detailed examination of the parametrisation of this stochastic search method postponed to Section 4.3.3.3. The amount of supervisory information N_{trd} is set to 2,000 data-points.

The experimental results are presented in Figures 4.3-4.4. In each case the validation dataset AUPRC is measured for each setting of α across a wide range of hashcode lengths (32-256 bits). Each point on the graphs is averaged over ten random training/validation/test splits of the dataset in accordance with the procedure outlined in Section 4.3.2. In Figure 4.3a the quantisation model is parametrised to use the vanilla $0/1$ codebook of Single Bit Quantisation (SBQ). It is clear that a setting of $\alpha = 1$ is preferred across all hashcode lengths for this particular instantiation of the quantisation model, with all weight being allocated to the supervised signal. Interestingly, the unsupervised signal is therefore deemed unreliable and not required. This finding is of significance to the future design of quantisation algorithms, which before the innova-

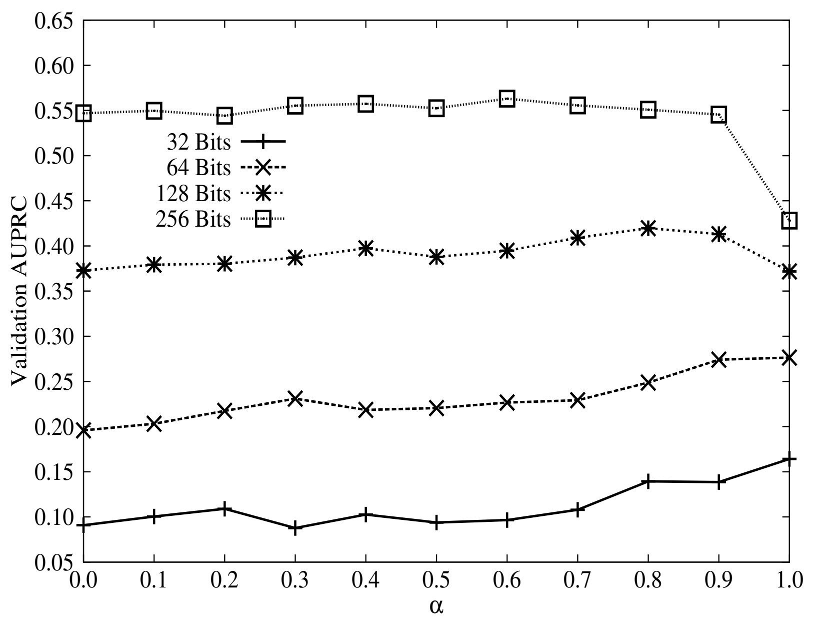
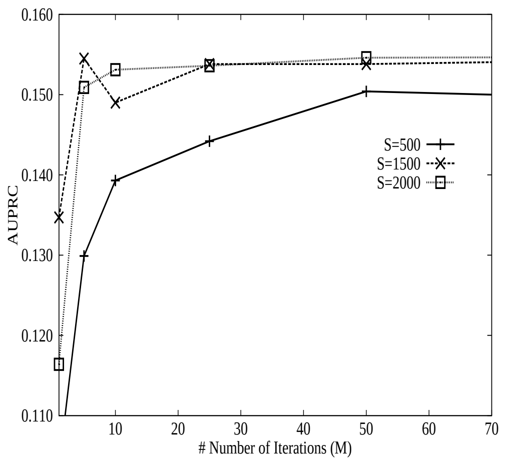


Figure 4.4: Variation in AUPRC with the value of α for the CIFAR-10 dataset and the MHQ codebook (LSH projections). $T = 3$ thresholds are optimised per projected dimension.

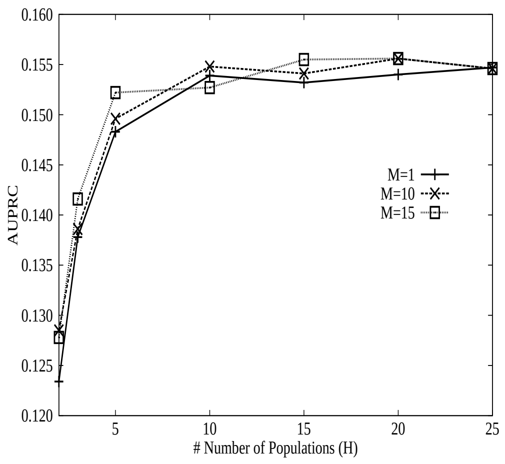
tion discussed in this chapter, all relied on the neighbourhood information arising from the unsupervised signal. A slightly different pattern emerges when the codebook is changed to the Double Bit Quantisation (DBQ) $00/11/10$ codebook (Figure 4.3b) and the Manhattan Hashing Quantisation (MHQ) natural binary codebook (NBC) (Figure 4.4). In both of these cases an $\alpha = 1$ is optimal for hashcodes of a lower length ($<$ 128 bits), while an $\alpha < 1$ leads to a higher retrieval effectiveness for longer hashcodes ($\geq$ 128 bits). This result suggests that for these non-standard codebooks (DBQ, MHQ) with LSH projections, in which more than one threshold is optimised per projected dimension, the influence of the unsupervised neighbourhood information resulting from the low-dimensional projection function becomes increasingly more important as the hashcode length increases.

4.3.3.3 Effect of Stochastic Search (Simulated Annealing versus Evolutionary Algorithms)

I discussed in Section 4.2.3 how two stochastic search methods, *evolutionary algorithms* and *simulated annealing*, can be used to efficiently optimise Equation 4.6 versus a purely brute-force search for the best threshold configuration. In this experiment I will compare both methods of stochastic search to see which is most effective for



(a) **Simulated Annealing**



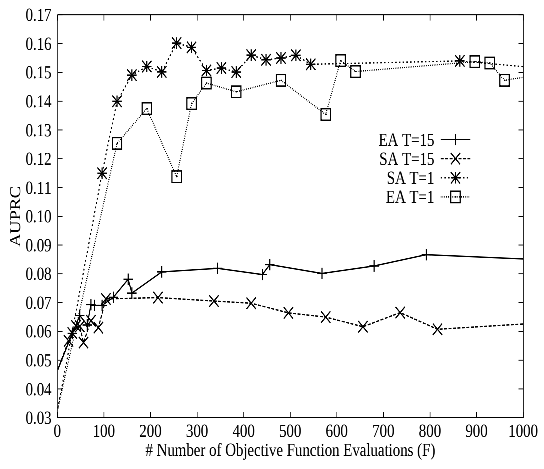
(b) **Evolutionary Algorithms**

Figure 4.5: Figure (a) shows the effect of initial temperature (S_0) and number of iterations (M) on the CIFAR-10 validation dataset AUPRC. Figure (b) illustrates the effect on validation dataset AUPRC of varying the number of populations (H) and the number of generations (M) for the evolutionary algorithm (EA). Results are for CIFAR-10 at 32 bits with LSH projections and $T = 1$ thresholds per projected dimension (0/1 codebook).

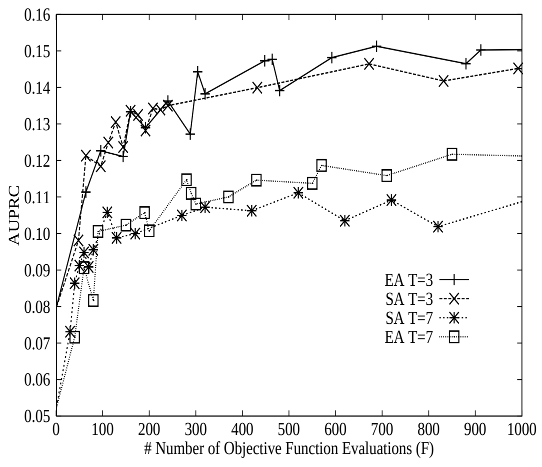
the task of threshold optimisation. Ideally we would like the stochastic search to find a threshold configuration that leads to the greatest AUPRC while taking a minimal number of evaluations of Equation 4.6.

Before comparing both stochastic search frameworks I firstly examine the initial temperature S_0 for simulated annealing since in preliminary experiments this single parameter was found to have the greatest effect on the final retrieval AUPRC. I anneal the temperature by $S = S_0 \times 0.95^m$ where S_0 is the initial temperature and S is the current temperature for iteration $m \in \{1, \ldots, M\}$. The next candidate threshold is selected with a step length that equals the temperature with the direction chosen uniformly at random. These simulated annealing settings were found to work best on a preliminary set of experiments. Figure 4.5a shows the effect on validation AUPRC of the temperature parameter with the number of iterations (M) of the simulated annealing stochastic search. A temperature of $S_0 = 2000$ and between $M = 30$-50 iterations appears to offer the fastest path to the highest AUPRC on the validation dataset. I therefore set $S_0 = 2000$ and constrain the number of iterations to below $M = 100$ for simulated annealing in the remaining experiments.

For evolutionary algorithms we have the mutation (ϕ), crossover (θ) and generation gap (G) parameters to set. In practice I find the default setting ($\phi = 0.001, \theta = 0.7,$



(a) $\mathbf{T=1,T=15}$



(b) $\mathbf{T=3,T=7}$

Figure 4.6: CIFAR-10 validation AUPRC for a certain number of objective function evaluations (F) for both simulated annealing (SA) and evolutionary algorithms (EA). Figure (a) shows the result for optimising 1 and 15 thresholds per projected dimension. Figure (b) shows the learning curves for 3 and 7 thresholds.

$G = 0.9$) in the Sheffield Genetic Algorithms Toolbox[14] to work well. I study in detail the effect of the remaining two parameters of the evolutionary algorithm, namely the number of populations (candidate threshold hypotheses) H and the number of generations (iterations) M. Figure 4.5b plots the AUPRC results arising from a grid search on the CIFAR-10 validation dataset for values of $M \in [1,\dots,15]$ and $H \in [1,\dots,15]$. A population of more than one ($H > 1$) appears to be critical for achieving the highest retrieval effectiveness, with the number of generations (M) having a smaller boost on the performance. For example, with just one generation and five populations the evolutionary algorithm attains an AUPRC of 0.1483 which is close to the value achieved with ten generations and five populations (0.1496 AUPRC). The fact that the AUPRC tails off rapidly for a low number of populations and generations ensures that the stochastic search remains efficient despite being an inherently randomised process.

The question arises as to which stochastic search method is better for the purposes of threshold learning. I plot in Figures 4.6a-4.6b the results of optimising 1,3,7 and 15 thresholds per projected dimension with simulated annealing (SA) and the evolutionary algorithm (EA). I note that for a single threshold (Figure 4.6a) it appears that SA reaches the highest validation AUPRC with a lower number of objective function evaluations than does EA. Nevertheless, they both reach a validation AUPRC

[14]`http://codem.group.shef.ac.uk/index.php/ga-toolbox`

that is approximately equal after a sufficient number of objective function evaluations ($F > 700$). For multiple thresholds per projected dimension (3,7,15) in Figures 4.6a-4.6b we see a different picture: here the EA reaches a higher validation AUPRC than SA. Furthermore, SA cannot reach the validation AUPRC achieved by EA even after $F = 1,000$. Based on these results I opt for evolutionary algorithms for the remainder of this thesis because this style of stochastic search appears to give a consistently good AUPRC across all threshold quantities. I set the number of generations to $M = 15$ and the number of individuals to $H = 15$. This setting of the parameters provides an acceptable tradeoff between effectiveness (final AUPRC achieved) and efficiency (time taken to learn the thresholds) for all three image collections. The experiments in this section were conducted on the CIFAR-10 dataset. In future work it would be prudent to confirm these results on additional datasets (NUS-WIDE, SIFT1M).

4.3.3.4 Evaluation of Training Time

The setting of the quantisation thresholds is a one-time offline training cost conducted prior to using the learnt thresholds to generate the hashcodes for the database and query data-points. Nevertheless, despite this being an offline process that will crucially not affect the nearest neighbour search query time, and therefore not affect the core reason for wanting to use hashing-based ANN search in the first place, it is imperative that the training cost be as low as possible so that the act of learning the hashcodes for large datasets remains tolerable. We have already seen in Section 4.3.3.1 how a relatively small, and sparse, adjacency matrix of around 1% of the total dataset size is sufficient for use in learning the quantisation thresholds. In this experiment I seek to measure how this offline processing cost compares against the baseline quantisation models using the optimal size N_{trd} of the supervisory adjacency matrix determined for the CIFAR-10 dataset in Section 4.3.3.1. I use broadly the same model configuration as I did in Section 4.3.3.1: namely the optimisation of thresholds for LSH-based projections.

The training timing results for the CIFAR-10 image dataset are shown in Table 4.6 using $N_{trd} = 2,000$. The training time of my multi-threshold quantisation model (NPQ) is an order of magnitude *faster* than the HQ algorithm of Liu et al. (2011) while being commensurate with the MHQ multi-threshold quantisation algorithm of Kong et al. (2012). This latter finding is particularly encouraging as the time complexity of NPQ is dependent on the square of the number of supervisory training data-points, whereas the k-means clustering algorithm used by MHQ has a linear dependence. In

	Number of Thresholds				
Model	**T = 1**	**T = 2**	**T = 3**	**T = 7**	**T = 15**
NPQ	0.180	0.202	0.225	0.360	0.639
SBQ	0.001	–	–	–	–
DBQ	–	0.002	–	–	–
MHQ	–	–	0.276	0.291	0.376
HQ	–	–	1.160	–	–

Table 4.6: Mean time taken (in seconds) *per projected dimension* to learn the quantisation thresholds with $N_{trd} = 2000$ on the CIFAR-10 dataset. The timing results were recorded on an otherwise idle Intel 2.7GHz, 16Gb RAM machine and averaged over ten random dataset partitions. All models are implemented in the same software stack (Matlab). The evolutionary algorithm had the setting $H = 15, M = 15$. NPQ timings include the time to compute the supervised and unsupervised terms in Equation 4.6.

practice, the computational time complexity of NPQ is substantially reduced by the sparsity of the matrix $\mathbf{S}$ and the efficient matrix operations that I use to enumerate the required true positive (TP), false positive (FP) and false negative (FN) counts.

To accelerate the training time of NPQ I avoid computing the indicator matrix $\mathbf{P}$ (Equation 4.1) and instead compute the F_1-measure by manipulating the highly sparse adjacency matrix $\mathbf{S}$. My more efficient counting procedure involves first rearranging (sorting) the rows and columns of the adjacency matrix $\mathbf{S}$ so that they are in the same order as the sorted projected values for projected dimension $\mathbf{y}^k$. This pre-processing step takes $O(N_{trd} \log N_{trd})$ time. Counting the TPs, FPs and FNs for each thresholded region then simply amounts to taking rectangular slices of the resulting matrix, which contain many fewer elements than the full adjacency matrix. Furthermore since this F_1-measure computation is repeatedly called by the stochastic search algorithm to evaluate candidate threshold positions, any computational savings in the corresponding function will positively impact the overall training time.

To better illustrate this more efficient method of counting the TPs, FPs and FNs we will consider the hypothetical projected dimension shown in Figure 4.7. In this diagram the data-points are arranged from left-to-right along the projected dimension in ascending order of their projected value. This means that data-point i has the lowest projected value while data-point d has the highest. As before, data-points with the same shape and colour are true nearest neighbours in the original higher dimensional

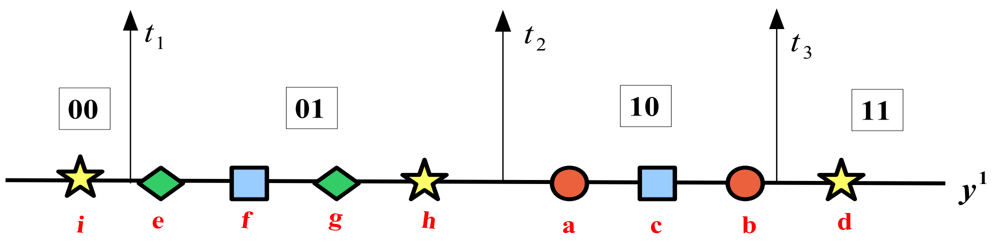


Figure 4.7: Example projected dimension used to illustrate an efficient method of computing the TPs, FPs and FNs required to compute the F_1-measure in Equation 4.6.

feature space. The corresponding adjacency matrix **S** encoding the pairwise relationships between the data-points is shown in matrix 4.9. For example, as data-points *a,b* are true nearest neighbours a '1' is placed in elements $S_{a,b}$ and $S_{b,a}$ of **S**. To efficiently compute the required TPs, FPs and FN counts I sort the rows and columns of **S** in the same order as the projected dimension in Figure 4.7. The rearranged adjacency matrix $\mathbf{S}^{'}$ is shown in matrix 4.10.

$$\begin{array}{c|ccccccccc} \mathbf{S} & a & b & c & d & e & f & g & h & i \\ \hline a & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ b & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ e & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ f & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ g & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ h & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ i & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{array} \tag{4.9}$$

With the adjacency matrix rearranged in this manner, determining the necessary TP, FP and FN counts for Equation 4.6 is possible entirely through efficient sparse matrix operations. These operations involve computing the number of 1's in $T+1$ rectangular slices of the adjacency matrix, with the size of the rectangular slices determined by the threshold positions. For our toy example presented in Figure 4.7, the four slices of $\mathbf{S}^{'}$ are shown in matrix 4.10. The number of true positives (TPs) is then simply half the number of 1's found in the four rectangular slices, which in this case is $TP = 4/2 = 2$. The number of FPs can be determined by firstly computing the total number of elements within the four rectangular slices (ignoring elements on the diagonal), which in this case is equal to $12+6=18$. Halving this value and subtracting the TPs, will give $FP = 18/2-2 = 7$. Finally the FNs are computed by subtracting the TP count from half the total number of non-zero elements[15] in $\mathbf{S}$. In this case $FN = 12/2-2 = 4$. We can confirm that these counts are indeed correct by appealing to Figure 4.7 and enumerating the counts manually. Importantly, none of these computations involves explicitly enumerating the zeroes in the matrix, enabling the required counts to be entirely determined from the sparse matrix representation alone, an important advantage in practice[16].

$$\begin{array}{c|ccccccccc}
\mathbf{S}' & i & e & f & g & h & a & c & b & d \\
\hline
i & \boxed{0} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
e & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
f & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
g & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
h & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
a & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
c & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
b & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
d & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \boxed{0}
\end{array} \tag{4.10}$$

4.3.3.5 Experiment I: Single Threshold Optimisation

In this experiment I examine the first hypothesis H_1 as outlined in Section 4.3.1. To investigate this hypothesis I optimise the position of a *single threshold* per projected dimension using Equation 4.6 and compare directly to standard Single Bit Quantisation (SBQ) which places the quantisation threshold directly at zero along the mean centered projected dimension. For relevant background information on SBQ please refer to Chapter 2, Section 2.5.1. I also present a random baseline (RND) which places a threshold uniformly at random along each projected dimension. Both baselines will make it obvious whether or not learning the threshold positions is in fact useful for improving retrieval effectiveness. To optimise the threshold I maximise Equation 4.6

[15]Counting the total number of 1's in the matrix need only be done once before starting the threshold optimisation, and reused in each objective function call.

[16]Given the matrix $\mathbf{S}$ is symmetric under the ε-NN groundtruth definition additional gains in efficiency can be realised by only storing the non-zero elements in the upper or lower triangular half of the matrix.

using stochastic search via *evolutionary algorithms*. The meta-parameter $\alpha \in [0,1]$ is set to 1.0 for all hashcode lengths, which was found to be optimal for the 0/1 codebook in our parameter study in Section 4.3.3.2. All factors of variation are kept the same between the three quantisation algorithms: specifically, I use the same codebook and the same ranking criterion to compute AUPRC (Hamming distance). I show the parametrisation of this experiment in Table 4.7 and the retrieval results in Tables 4.8-4.9.

Method	**# Thresholds/dim**	**Encoding**	**Ranking Strategy**
NPQ	1	0/1	Hamming
SBQ	1	0/1	Hamming

Table 4.7: Parametrisation of the quantisation methods studied in Experiment I.

(a) Quantising LSH projections with $T = 1$ threshold per projected dimension

I firstly examine the retrieval effectiveness arising from the quantisation of LSH projections with a learnt threshold. The experimental results in Table 4.8 suggest that placing a threshold at zero is a sub-optimal quantisation strategy. Retrieval effectiveness is significantly lower (Wilcoxon signed rank test, $p < 0.01$) than optimising the threshold using my semi-supervised quantisation algorithm (NPQ). Figure 4.8a shows that the superior performance of my quantisation algorithm holds across a wide range of hashcode lengths on the CIFAR-10 dataset. In the case of LSH, these results confirm the claim set out in Chapter 2, Section 2.5.1 that a threshold set by default at zero is very likely to divide many related data-points on opposite sides of the quantisation threshold. This finding is a particularly encouraging result for two reasons: firstly, it supports the case for further studying the threshold optimisation problem in the context of ANN search. Secondly, as LSH is an undoubtedly popular method for hashing-based ANN search, replacing SBQ with my threshold optimisation algorithm (NPQ) is likely to give an immediate boost in retrieval effectiveness on the many end-applications that rely on this hash function.

(b) Quantising PCA projections with $T = 1$ threshold per projected dimension

The NPQ quantisation algorithm is independent of the projection stage and therefore has the appealing advantage of being applicable to the projections arising from

	CIFAR-10	**NUS-WIDE**	**SIFT1M**
LSH + NPQ	**0.1963** (0.1899)▲▲	**0.5008** (0.5006)▲▲	**0.1220** (0.1297)▲▲
LSH + SBQ	0.1069 (0.1068)	0.3395 (0.3392)	0.0974 (0.0974)
LSH + RND	0.0339 (0.0339)	0.0253 (0.0252)	0.0103 (0.0103)

Table 4.8: AUPRC for the single threshold ($T = 1$) optimisation experiment at 32 bits for LSH projections ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over SBQ. The improved splitting strategy result are in brackets.

	CIFAR-10	**NUS-WIDE**	**SIFT1M**
PCA + NPQ	**0.1018** (0.1012)▲▲	**0.1208** (0.1205)▲▲	**0.2085** (0.2085)▲▲
PCA + SBQ	0.0387 (0.0388)	0.0477 (0.0477)	0.1081 (0.1081)
PCA + RND	0.0297 (0.0297)	0.0075 (0.0075)	0.0148 (0.0148)

Table 4.9: AUPRC for the single threshold ($T = 1$) optimisation experiment at 32 bits for PCA projections. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over SBQ. The improved splitting strategy result are in brackets.

any hash function. As a consequence I also study the quantisation of PCA-based projections given the centrality of PCA to many of the data-dependent (unsupervised) projection functions in the literature (Chapter 2, Section 2.6.3). The retrieval results for this projection are presented in Table 4.9. The first point of note with these results, when comparing to the LSH retrieval results in Table 4.9, is the substantially lower effectiveness of PCA-based projections versus LSH projections for nearest neighbour search on two out of the three datasets (CIFAR-10 and NUS-WIDE). For example, on CIFAR-10 at 32 bits LSH+SBQ realises a 176% relative increase in AUPRC versus PCA+SBQ. This result suggests that PCA projections, despite their data-dependent nature, are less effective for partitioning the input space compared to randomly drawn LSH hyperplanes. I hypothesise that this drop in retrieval effectiveness is related to the *imbalanced variance problem* discussed in Chapter 2, Section 2.6.3.1. The eigenvectors with the lowest eigenvalues are generally unreliable and provide little information on the input feature space. This means that any hashcode bits generated from these eigenvectors are ineffective for distinguishing related and unrelated data-points. I introduce quantisation algorithms that address the imbalanced variance problem in Chapter 5.

In Table 4.9 it is apparent that optimising a single threshold (PCA+NPQ) for PCA

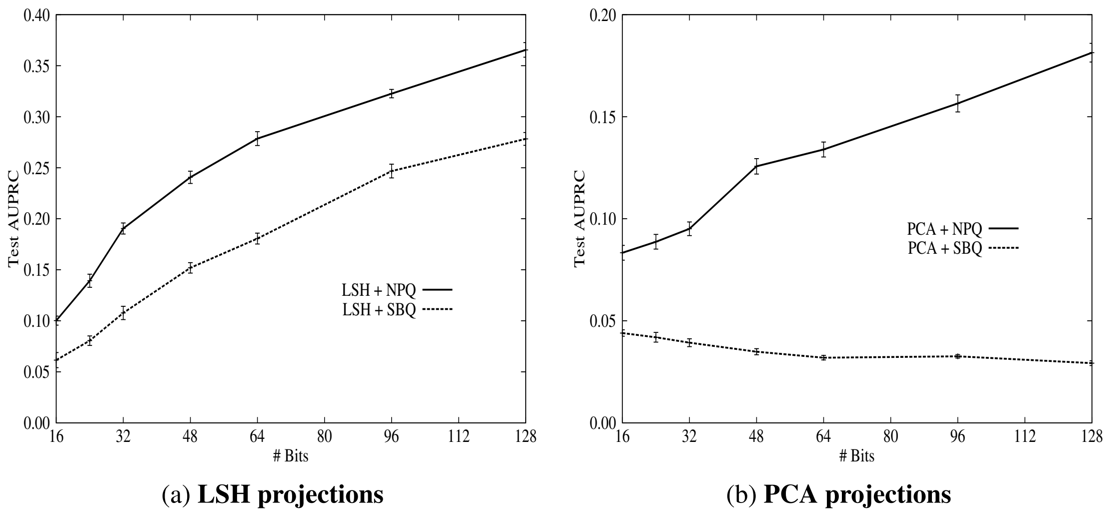


Figure 4.8: AUPRC versus hashcode length on CIFAR-10 for Experiment I. Results for LSH projections are shown in Figure (a) and PCA projections in Figure (b). The bars show the standard error of the mean.

projections yields a significantly higher effectiveness than either a statically placed threshold (PCA+SBQ) or a random threshold placement (PCA+RND). This improvement in retrieval effectiveness is confirmed across a wide range of hashcode lengths on the CIFAR dataset (Figure 4.8b) where the difference in AUPRC continues to increase as the hashcode length is increased. This result suggests that optimising a single threshold is also beneficial for PCA projections. Given the generality of the multi-threshold quantisation algorithm to PCA projections we have reason to suspect it may provide an additional boost in retrieval effectiveness when used to quantise projections from other data-dependent projection functions. I defer examination of this hypothesis to Section 4.3.3.8.

In contrast, in Figure 4.8b, it is readily apparent that the retrieval effectiveness of PCA+SBQ declines as the hashcode length increases. This effect can be attributed to the noisy (low variance) PCA dimensions that are being used to generate the increasingly longer hashcodes. Unlike in standard vision algorithms, such as those for image annotation (Moran and Lavrenko (2014)), the dimensions can be weighted so to emphasise their importance (or not) to the specific task. In hashing, however, there is no such notion of a dimension weighting and all dimensions are therefore treated equally when computing and ranking with the Hamming distance. This equal treatment in the presence of noisy dimensions can markedly affect performance as is observed for PCA+SBQ in Figure 4.8b. In Chapter 5, I explore how a notion of dimension weighting through multiple bit assignment can mitigate this particular issue in the context of hashing.

(c) Standard dataset splitting strategy versus the improved splitting strategy

The final observation I make from the experimental results in Tables 4.8-4.9 is the similarity of the AUPRC arising from the standard splitting strategy (Chapter 3, Section 3.5.1) used in the learning to hash literature and the improved strategy outlined in Chapter 3, Section 3.5.2. In many cases there is *no significant difference* in the AUPRC achieved when computing retrieval results using either strategy. I previously made the argument in Chapter 3, Section 3.5 that the literature standard method of defining test and training splits ran the risk of overfitting the hash functions to the training dataset. The reason for this assumption was that the training dataset used to learn the hash functions was also a subset of the database used for the test retrieval run. The results in this section downplay this fear as I find that holding out a completely separate test database for the final retrieval run leads to not only the same ranking of the quantisation algorithms in terms of most effective to least effective but also nearly identical AUPRC as averaged over ten random dataset splits. Despite the literature standard splitting strategy arguably being less technically sound from a machine learning perspective I provide the first evidence here that it is nevertheless a valid evaluation methodology. The retrieval results arising from the literature standard splitting strategy will only be provided in the remaining experiments of this chapter.

4.3.3.6 Experiment II: Double Threshold Optimisation

In this experiment I will examine the hypothesis (H_2) that optimising *two* thresholds per projected dimension with my proposed semi-supervised objective function (Equation 4.6) can achieve a higher retrieval effectiveness than the Double Bit Quantisation (DBQ) algorithm of Kong and Li (2012a). The DBQ algorithm was outlined in detail in Chapter 2, Section 2.5.3. As two thresholds will induce three regions along the projected dimension we can no longer uniquely label each region using a single bit encoding as I did for Experiment I in Section 4.3.3.5. I therefore opt for the DBQ multi-bit codebook in which two bits are assigned per projected dimension. This encoding scheme is shown in Figure 2.11 of Chapter 2. In addition to DBQ and the purely random threshold setting baseline RND, I also compare to the baseline EQL which sets the thresholds at equally spaced intervals along the projected dimension. If

Method	**# Thresholds/dim**	**Encoding**	**Ranking Strategy**
NPQ	2	01/11/10	Hamming
DBQ	2	01/11/10	Hamming
RND	2	01/11/10	Hamming
EQL	2	01/11/10	Hamming
SBQ	1	0/1	Hamming

Table 4.10: Parametrisation of the quantisation methods studied in experiment II

we denote as $T = 2$ the number of thresholds per projected dimension, then the width of each interval w is computed as specified in Equation 4.11

$$w = \frac{y^k_{max} - y^k_{min}}{T+1} \tag{4.11}$$

where $y^k_{min} \in \mathbb{R}$ and $y^k_{max} \in \mathbb{R}$ are the minimum and maximum projected values of projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$. Given the width $w \in \mathbb{R}$ the quantisation thresholds are then placed at the positions along the projected dimension given by $y^k_{min} + w$, $y^k_{min} + 2w$, ..., $y^k_{min} + Tw$. The parametrisation of the quantisation algorithms studied in this experiment is shown in Table 4.10. The NPQ, DBQ, RND and EQL quantisation algorithms all assign 2 bits per projected dimension and therefore only use $K/2$ of the number of available hyperplanes to generate K bits. In the case of LSH projections the first $K/2$ hyperplanes are used to partition the input space, while for PCA projections the $K/2$ hyperplanes with the largest eigenvalues are used because these are generally more reliable (Liu et al. (2011)). As the SBQ algorithm only assigns 1 bit per projected dimension it will use all K available hyperplanes to generate K bits. I mirror the experimental evaluation in Section 4.3.3.5 by analysing the retrieval results with both LSH and PCA projections. The interpolation parameter $\alpha \in [0, 1]$ in Equation 4.6 is set to $\alpha = 1.0$ for hashcodes of length $K < 128$ bits and $\alpha = 0.8$ for $K \geq 128$ bits. These settings of α were found to be optimal in the parameter study conducted in Section 4.3.3.2.

(a) Quantising LSH projections with $T = 2$ thresholds per projected dimension

The retrieval results obtained by quantising LSH projections and using the resulting hashcodes for nearest neighbour search are presented in Table 4.11 for a hashcode length of 32 bits and in Figure 4.9 for hashcodes of length 16 through to 128 bits.

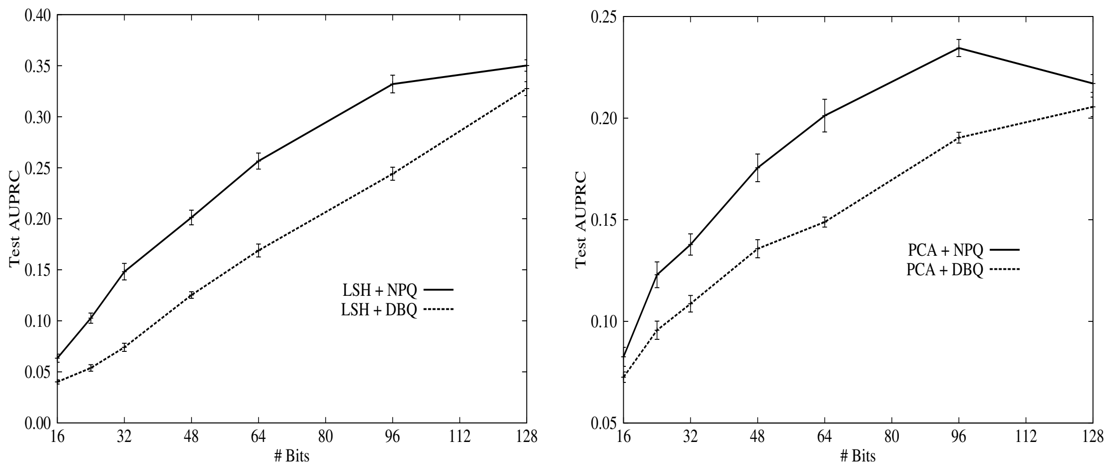
	CIFAR-10	**NUS-WIDE**	**SIFT1M**
LSH + NPQ	**0.1535▲▲**	**0.3459**	0.0933
LSH + DBQ	0.0786	0.1460	0.0764
LSH + SBQ	0.1069	0.3395	**0.0974**
LSH + EQL	0.0342	0.0228	0.0208
LSH + RND	0.0319	0.0188	0.0062

Table 4.11: AUPRC for the double threshold ($T = 2$) quantisation experiment (II) for LSH projections. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over SBQ.

Examining the results in Table 4.11 we observe that the multi-threshold optimisation model (LSH+NPQ) attains a statistically significant increase (Wilcoxon signed rank, $p < 0.01$) in retrieval effectiveness with respect to the DBQ, EQL and RND baseline quantisation algorithms across all three still image collections. For example, on the CIFAR-10 dataset at 32 bits LSH+NPQ achieves a 95% relative increase in AUPRC versus LSH+DBQ. This result suggests that, for LSH projections, the quantisation algorithm proposed in this chapter is significantly more effective at the placement of two thresholds per projected dimension in comparison to the DBQ quantisation algorithm.

I observe mixed results when comparing the retrieval effectiveness of LSH+SBQ versus my own quantisation model (LSH+NPQ). Recall from Chapter 2, Section 2.5.1 that SBQ assigns one bit per projected dimension with a single threshold placed at zero along each projected dimension. On the CIFAR-10 dataset optimising two thresholds per projected dimension using my quantisation model yields a significant 44% relative increase in AUPRC versus statically placing a single threshold per projected dimension (LSH+SBQ). Surprisingly the DBQ quantisation model (LSH+DBQ) obtains a significantly lower retrieval effectiveness than a single threshold quantisation (LSH+SBQ) on the same dataset. This result again suggests that the DBQ threshold optimisation algorithm is not as effective as my own. However, I note an opposite trend on the larger NUS-WIDE and SIFT1M datasets where there is no significant difference between a single threshold at zero (LSH+SBQ) and allocating and optimising two thresholds per projected dimension (LSH+NPQ). The most likely reason for this observation is the use of half the number of hyperplanes, compared to all hyperplanes in the case of a single threshold per projected dimension.

The question arises as to whether or not it is actually beneficial to assign and opti-



(a) **LSH projections** (b) **PCA projections**

Figure 4.9: AUPRC versus hashcode length on CIFAR-10 for Experiment II. Results for LSH projections are shown in Figure (a) and PCA projections in Figure (b). The bars show the standard error of the mean.

mise two thresholds per projected dimension for LSH projections. We can answer this question by comparing the retrieval results in Table 4.8 to the retrieval results in Table 4.11. Recall from Section 4.3.3.5 that the retrieval results in Table 4.8 for LSH+NPQ were obtained by optimising a *single* threshold per projected dimension. For example on the CIFAR-10 dataset optimising a *single threshold* per projected dimension using my quantisation model obtains an AUPRC of 0.1963 (Table 4.8). This retrieval result should be compared to an AUPRC of 0.1535 which is obtained through optimising two thresholds per projected dimension (Table 4.11) with the same model. A similar pattern is also observed on the NUSWIDE and SIFT1M image collections. We can conclude that for LSH allocating and optimising two thresholds per projected dimension is a much less effective quantisation approach compared to optimising a single threshold per projected dimension, with the caveat that we are using the Hamming distance for hashcode comparison. In Section 4.3.3.8, I will show that using multiple thresholds can be more effective than a single threshold for LSH when the Manhattan distance is used for hashcode ranking in the manner proposed by Kong et al. (2012).

In all experiments in this chapter, the strategy taken is to reduce the quantity of hyperplanes in proportion to the number of bits added. For example, if $B = 2$ bits are assigned per projected dimension, as was the case for the experiments in this section, then the quantity of hyperplanes is halved so that the total number of bits used does not exceed the bit budget K. For example, for an assigned bit budget of 32 bits, only

	CIFAR-10	**NUS-WIDE**	**SIFT1M**
PCA + NPQ	**0.1388▲▲**	**0.1526▲▲**	**0.2478▲▲**
PCA + DBQ	0.1084	0.0723	0.1816
PCA + SBQ	0.0387	0.0477	0.1081
PCA + EQL	0.0879	0.0221	0.0495
PCA + RND	0.0363	0.0112	0.0139

Table 4.12: AUPRC for the double threshold ($T = 2$) quantisation experiment (II) for PCA projections. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over DBQ.

16 hyperplanes will be used with an allocation of $B = 2$ bits per hyperplane. The main reason that this strategy is used here and is prevalent in the literature (Kong et al. (2012); Kong and Li (2012a)) is to ensure that all models use the same computational resources for hashcode storage and comparison. Permitting the multi-bit quantisation models to use the same number of planes but with more bits would involve using more storage and computation time for their hashcodes (as they will consist of more bits). The focus in the hashing literature is to maximise performance with respect to a fixed bit budget K, which has the desirable effect of constraining the computational resources used. The alternate strategy of increasing the number of bits for a fixed *hyperplane* budget would likely see an increase in effectiveness as more bits are added. For example, in Figure 4.8a assigning 1 bit per hyperplane for 48 hyperplanes achieves an AUPRC of 0.2406, whereas assigning 2 bits per hyperplane for 48 hyperplanes in Figure 4.9a attains a substantially higher AUPRC of 0.3320.

(b) Quantising PCA projections with $T = 2$ thresholds per projected dimension

I present in Table 4.12 the retrieval results arising from allocating and optimising two thresholds per projected dimension for PCA projections. In stark contrast to LSH projections I find that optimising two thresholds per PCA projected dimension results in significantly higher (Wilcoxon signed rank, $p < 0.01$) retrieval effectiveness compared to a single threshold (PCA+SBQ) across all three datasets. For example on the CIFAR-10 dataset at 32 bits PCA+NPQ with two thresholds per projected dimension attains a 259% relative increase in AUPRC versus a single threshold quantisation (PCA+SBQ). Similar increases in retrieval effectiveness are also observed on the NUS-WIDE and SIFT1M datasets. This result suggests that allocating more than

Method	# Thresholds/dim	Encoding	Ranking Strategy
NPQ	3,7,15	NBC	Manhattan
MHQ	3,7,15	NBC	Manhattan
RND	3,7,15	NBC	Manhattan
EQL	3,7,15	NBC	Manhattan

Table 4.13: Parameterisation of the quantisation methods studied in experiment III

one threshold (two in this experiment) to the eigenvectors (hyperplane normal vectors) with the greatest eigenvalues directly benefits retrieval effectiveness, a finding which I explore in more detail in Chapter 5. This finding is corroborated by Table 4.9 in Section 4.3.3.5 in which optimising a single threshold with NPQ results in a lower retrieval effectiveness compared to the optimisation of two thresholds using my model. Despite the higher retrieval effectiveness for PCA+NPQ with two thresholds per projected dimension it still cannot match the retrieval performance of LSH+NPQ with one threshold per projected dimension (Table 4.8), at least for two of the considered datasets (CIFAR-10, NUS-WIDE).

Finally, I note that my multi-threshold quantisation model (PCA+NPQ) also demonstrates a higher retrieval effectiveness than the DBQ quantisation algorithm of Kong and Li (2012a) (PCA+DBQ). This result provides further evidence as to the importance of fusing together two complementary signals in the form of affinity information between the data-points in the input feature space (provided in the adjacency matrix $\mathbf{S}$) and information captured by the low-dimensional projection function (i.e. PCA) itself.

4.3.3.7 Experiment III: Multiple ($T = 3, 7, 15$) Threshold Optimisation

The experiments in this section compare the multi-threshold quantisation algorithm (NPQ) introduced in this chapter against the Manhattan Hashing Quantisation (MHQ) algorithm of Kong et al. (2012) and reviewed in Chapter 2, Section 2.5.4. In doing so I seek to answer hypothesis H_3 as to whether or not optimising *three or more* thresholds with the semi-supervised objective function (Equation 4.6) can yield a higher retrieval effectiveness than MHQ. To the best of my knowledge MHQ is the only quantisation algorithm for hashing-based ANN search that generalises to 3+ thresholds per projected dimension, a feat that is possible through the use of Natural Binary Code (NBC) to encode the thresholded regions and Manhattan distance to compute the distances between the hashcodes. For a full discussion of MHQ please refer to Chapter 2, Section

	# Thresholds		
Method	**3**	**7**	**15**
LSH + NPQ	**0.1621▲▲**	**0.0921▲**	**0.0830▲▲**
LSH + MHQ	0.0877	0.0645	0.0517
LSH + EQL	0.0617	0.0564	0.0503
LSH + RND	0.0357	0.0339	0.0384

Table 4.14: AUPRC on the CIFAR-10 dataset for LSH projections with a hashcode length of 32 bits and varying thresholds ($T = 3, 7, 15$). ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over MHQ. ▲/▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.05$) over MHQ.

2.5.4.

The parametrisation of this experiment is shown in Table 4.13. I configure my multi-threshold quantisation model to use the MHQ codebook and seek to optimise 3, 7 and 15 thresholds per projected dimension which is equivalent to an assignment of 2, 3 and 4 bits for each hyperplane respectively. In each case to generate K bits $\lfloor K/B \rfloor$ hyperplanes are needed, where $T = 2^B - 1$: for example, to generate 32 bits with an assignment of 3 thresholds per projected dimension only $\lfloor 32/2 \rfloor = 16$ of the available hyperplanes are used. As for Section 4.3.3.5 and Section 4.3.3.6 the first $\lfloor K/B \rfloor$ LSH hyperplanes are selected, while the $\lfloor K/B \rfloor$ PCA hyperplanes capturing the highest variance in the input feature space are used. The meta-parameter $\alpha \in [0, 1]$ in Equation 4.6 is set to $\alpha = 1.0$ for hashcodes of length $K < 128$ bits and $\alpha = 0.8$ for $K \geq 128$ bits, as was found to be optimal in the parameter study conducted in Section 4.3.3.2.

(a) Quantising LSH projections with $T = 3, 7, 15$ thresholds per projected dimension

The retrieval results for this experiment are presented in Table 4.14 and Figure 4.10a for LSH projections. I confirm hypothesis H_3 for LSH projections given the significantly higher AUPRC (Wilcoxon signed rank test ($p < 0.01$)) for LSH+NPQ versus the baseline quantisation algorithms, and in particular LSH+MHQ. This strongly suggests that the quantisation algorithm introduced in this chapter achieves state-of-the-art retrieval effectiveness for nearest neighbour search using *any* quantity of thresholds. Furthermore, it is abundantly clear that LSH has a preference for a lower number of

	# Thresholds		
Method	**3**	**7**	**15**
PCA + NPQ	**0.1660▲**	**0.1824▲▲**	**0.1504▲**
PCA + MHQ	0.1408	0.1456	0.1299
PCA + EQL	0.0865	0.1236	0.1216
PCA + RND	0.0471	0.0708	0.0780

Table 4.15: AUPRC on the CIFAR-10 dataset for PCA projections with a hashcode length of 32 bits and varying thresholds ($T = 3, 7, 15$). ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over MHQ. ▲/▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.05$) over MHQ.

thresholds ($T = 3$) with AUPRC falling as more thresholds ($T = 7, 15$) are allocated to each projected dimension. This result is true not only for my quantisation model but also for the multi-threshold baseline quantisation algorithms (MHQ, EQL). This finding accords with earlier observations made in Section 4.3.3.5 and Section 4.3.3.6 in which I discovered that optimising just a single threshold is the best strategy for LSH. Indeed, by comparing Table 4.8 to Table 4.14 it is apparent that optimising a single threshold yields the highest overall AUPRC for LSH (namely 0.1963 AUPRC versus an AUPRC of 0.1621).

(b) Quantising PCA projections with $T = 3, 7, 15$ thresholds per projected dimension

I again confirm hypothesis H_3 by examining the AUPRC for PCA projection quantisation in Table 4.15 and Figure 4.10b. There is a statistically significant (Wilcoxon signed rank, $p < 0.01$) increase in AUPRC when comparing PCA+NPQ to PCA+MHQ for a hashcode length of 32 bits. I also find that PCA+NPQ dominates PCA+MHQ for hashcode lengths of between 16-128 bits (Figure 4.10b). This result provides further evidence regarding the effectiveness of positioning multiple quantisation thresholds by optimising Equation 4.6 using evolutionary algorithms in contrast to a purely unsupervised (k-means) clustering of the projected dimension. In contrast to LSH projections I again find that multiple thresholds lead to the highest retrieval effectiveness (Table 4.15) for PCA projections. For the CIFAR-10 dataset, seven thresholds per projected dimension, which equates to three bits per hyperplane, appears to be optimal with a higher ($T = 15$) or lower ($T = 3$) number of thresholds leading to a significantly lower

Method	# Thresholds/dim	Encoding	Ranking Strategy
NPQ	3	NBC	Manhattan
MHQ	3	NBC	Manhattan
EQL	3	NBC	Manhattan
HQ	1+3	0/1	Hamming
SBQ	1	0/1	Hamming

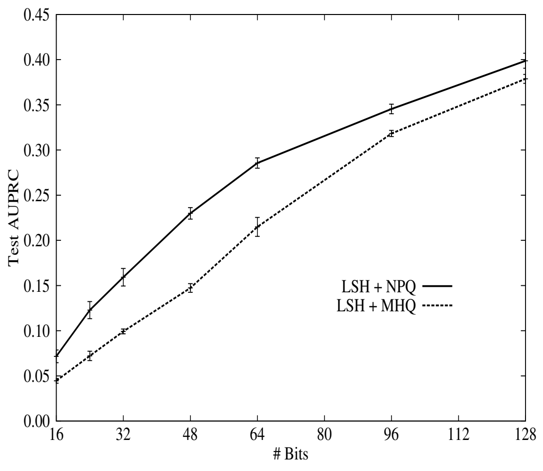
Table 4.16: Parametrisation of the quantisation methods studied in experiment IV. NBC stands for natural binary code.

AUPRC. This observation suggests that there is a *sweet spot for the threshold allocation* per projected dimension, an important finding that I will investigate and expand upon much further in the next chapter of this dissertation. I can furthermore measure the efficacy of the MHQ codebook and pairwise comparison metric (Manhattan distance) in comparison to the DBQ (00/11/10) codebook and the vanilla binary (0/1) codebook with Hamming distance. The maximum AUPRC achieved with PCA+NPQ is 0.1824 using the MHQ codebook and the Manhattan ranking strategy (Table 4.15). This should be contrasted with 0.1388 AUPRC for the DBQ codebook (Table 4.12) and 0.1018 AUPRC for the binary codebook (Table 4.9). Clearly the MHQ codebook and the Manhattan ranking strategy for pairwise hashcode comparison leads to the highest AUPRC and therefore is clearly more effective than either alternative. I therefore confirm the original findings of Kong et al. (2012) when using their codebook and ranking strategy with my own quantisation model.

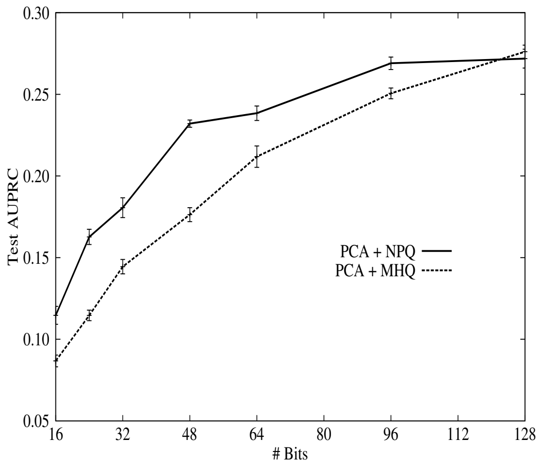
4.3.3.8 Experiment IV: Generalisation to other Projection Functions

In this final experiment I will expand the number of projection functions to be quantised from LSH and PCA to other more recent data-dependent models, namely Spectral Hashing (SH), Iterative Quantisation (ITQ) and Shift Invariant Kernel Hashing (SKLSH). SH was discussed in detail in Chapter 2, Section 2.6.3.2, while ITQ was reviewed in Chapter 2, Section 2.6.3.3 and SKLSH in Chapter 2, Section 2.6.2.1[17]. The experimental setup is shown in Table 4.16. The ability of my multi-threshold quantisation algorithm to generalise to other projection functions is measured against six quantisation model baselines: the data-dependent models MHQ, DBQ, and HQ and

[17] I use an SKLSH kernel bandwidth $\gamma = 1$ for CIFAR-10 and NUS-WIDE and $\gamma = 0.000001$ for SIFT1M which allows me to replicate the MHQ results reported by Kong et al. (2012).



(a) **LSH projections**



(b) **PCA projections**

Figure 4.10: AUPRC on the CIFAR-10 dataset for Experiment III ($T = 3$ thresholds per projected dimension). Results for LSH projections are shown in Figure (a) and PCA projections in Figure (b). The bars show the standard error of the mean.

the data-independent quantisation models SBQ, RND and EQL.

In all cases each quantisation algorithm is configured to generate hashcodes of length 32 bits for the CIFAR-10, NUS-WIDE and SIFT1M datasets, and the retrieval effectiveness of those hashcodes measured using the Hamming ranking evaluation paradigm and the AUPRC metric. I parametrise my own multi-threshold quantisation algorithm with the MHQ codebook and Manhattan ranking strategy as was the case for hypothesis H_3 in Section 4.3.3.7, setting 3 thresholds (or 2 bits) per projected dimension. Three thresholds (equivalently 2 bits) per projected dimension was in general found to work the best for MHQ across a wide selection of projection functions in the original paper (Kong et al. (2012)). The interpolation parameter $\alpha \in [0,1]$ in Equation 4.6 is set to $\alpha = 1.0$ for hashcodes of length $K < 128$ bits and $\alpha = 0.8$ for $K \geq 128$ bits. This configuration was found to be optimal in the experiments in Section 4.3.3.2. The hierarchical quantisation (HQ) algorithm is tied to the anchor graph hashing (AGH) projection function as was discussed in Chapter 2, Section 2.6.3.4. I therefore use the default AGH parameters of 300 anchor data-points and 5 nearest anchors as suggested in Liu et al. (2011).

(a) Generalisation to other Projection Functions

The results of this experiment are shown for CIFAR-10, NUS-WIDE and SIFT1M image datasets in Tables 4.17-4.19. In each table the projections resulting from the projection functions listed along the first column are quantised into binary hashcodes by applying the quantisation algorithms detailed along the top row. The results listed are for a hashcode length of 32 bits, although I find that the higher retrieval effectiveness of my multi-threshold quantisation model persists for longer and shorter hashcode lengths. It is immediately obvious that my own multi-threshold quantisation algorithm significantly (Wilcoxon signed rank test, $p < 0.01$ or $p < 0.05$) outperforms the *six baselines* quantisation schemes across *all five* different projection functions and on *all three* image datasets. This is a strong result that clearly shows the generality and power of the proposed multi-threshold quantisation model. The key difference between my proposed model (NPQ) and the state-of-the-art quantisation models, DBQ and MHQ, is that the latter rely entirely on the unsupervised signal arising from the low-dimensional projection function that is used, such as ITQ or LSH. The findings in this section suggest that these projection methods are somewhat limited in their ability to preserve the neighbourhood information between the data-points in the low-dimensional projected space. Quantising the resulting projections using an unsupervised one-dimensional clustering algorithm such as k-means (in the case of MHQ) is therefore sub-optimal. The quantisation model, and therefore the associated clustering algorithm, needs to take into account supervised information resulting from the original high-dimensional feature space.

The boost in retrieval effectiveness is particularly encouraging for ITQ projections, which is widely considered to be a state-of-the-art data-dependent (unsupervised) projection function. ITQ quantised with SBQ yields the highest retrieval effectiveness compared to any of the other considered projection functions quantised with SBQ. Replacing SBQ with my own multi-threshold quantisation algorithm (NPQ) and quantising the resulting projections yields a further increase in retrieval effectiveness for ITQ: from a 47% relative rise in AUPRC on the CIFAR-10 dataset to a 92% increase for SIFT1M. I note that the other data-dependent quantisation models (DBQ, MHQ) have a *lower* AUPRC when quantising ITQ projections on the CIFAR-10 and NUS-WIDE datasets compared to SBQ. This suggests that my own multi-threshold quantisation algorithm is the only data-dependent quantisation model that attains consistently better performance for ITQ across different image datasets. A selection of qualitative results comparing the top ten ranked retrieval effectiveness of ITQ+NPQ and ITQ+SBQ are displayed in Tables 4.20-4.23.

The experimental results in Tables 4.17-4.19, combined with the findings of the previous experiments (Sections 4.3.3.5-4.3.3.7), indicate that optimising the semi-

	Quantisation Model						
Projection	**NPQ**	**SBQ**	**MHQ**	**DBQ**	**HQ**	**EQL**	**RND**
LSH	**0.1621▲▲**	0.0954	0.0877	0.0850	–	0.0617	0.0357
ITQ	**0.3917▲▲**	0.2669	0.2168	0.2205	–	0.1182	0.0708
SH	**0.1834▲▲**	0.0626	0.1365	0.0952	–	0.1172	0.0588
PCA	**0.1660▲▲**	0.0387	0.1408	0.1084	0.0775	0.0865	0.0471
SKLSH	**0.1063▲▲**	0.0513	0.0610	0.0418	–	0.0517	0.0407

Table 4.17: AUPRC on the CIFAR-10 dataset with a hashcode length of 32 bits. The quantisation algorithms listed on the first row are used to quantise the projections from the hash functions in the first column. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over MHQ or SBQ, whichever baseline has the highest AUPRC.

	Quantisation Model						
Projection	**NPQ**	**SBQ**	**MHQ**	**DBQ**	**HQ**	**EQL**	**RND**
LSH	**0.4238▲**	0.3395	0.1551	0.1501	–	0.0491	0.0292
ITQ	**0.5130▲**	0.4842	0.3140	0.3960	–	0.0115	0.0235
SH	**0.1965▲▲**	0.0232	0.0708	0.0337	–	0.0380	0.0245
PCA	**0.2178▲▲**	0.0477	0.0734	0.0809	0.0491	0.0186	0.0126
SKLSH	**0.2650▲▲**	0.0310	0.0515	0.0270	–	0.0356	0.0242

Table 4.18: AUPRC on the NUS-WIDE dataset with a hashcode length of 32 bits. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over MHQ or SBQ, whichever baseline has the highest AUPRC. ▲/▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.05$) over MHQ or SBQ.

supervised objective in Equation 4.6 is a superior method of learning the quantisation thresholds for nearest neighbour search compared to fully unsupervised techniques such as k-means clustering (MHQ) or spectral graph partitioning (HQ). I therefore confirm the final hypothesis of this chapter H_4 by showing the generality of my algorithm to a wide selection of projection functions, both data-independent and data-dependent. This is in addition to the generality of the algorithm to other codebooks and to different quantities of quantisation threshold per projected dimension as was confirmed in the experiments in Sections 4.3.3.5-4.3.3.7. To the best of my knowledge the multi-threshold quantisation model introduced in this chapter is the first scalar quantisation

	Quantisation Model						
Projection	**NPQ**	**SBQ**	**MHQ**	**DBQ**	**HQ**	**EQL**	**RND**
LSH	**0.1339▲▲**	0.0974	0.1076	0.0772	–	0.0449	0.0178
ITQ	**0.3190▲▲**	0.1664	0.2699	0.1999	–	0.0881	0.0317
SH	**0.3269▲▲**	0.1141	0.2635	0.1413	–	0.2235	0.0709
PCA	**0.3332▲▲**	0.1093	0.2540	0.1679	0.0605	0.1217	0.0359
SKLSH	**0.1066▲▲**	0.0070	0.0714	0.0200	–	0.0229	0.0148

Table 4.19: AUPRC on the SIFT1M dataset with a hashcode length of 32 bits. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$), whichever baseline has the highest AUPRC.

model for nearest neighbour search that leverages a supervisory signal for threshold placement. Given the impressive gains in retrieval effectiveness I believe there to be a fruitful research avenue in investigating new fully supervised or semi-supervised scalar-quantisation models for hashing.

4.4 Conclusions

In this chapter a new quantisation algorithm was introduced for converting real-valued projections resulting from a low-dimensional projection function into binary hashcodes for the purpose of nearest neighbour search. The quantisation algorithm extended the popular and widely used single bit quantisation (SBQ) method in two important directions: firstly, one or more thresholds were permitted per projected dimension, instead of the limiting assumption of SBQ in which only a single threshold is allocated; and secondly, the position of the threshold(s) along each projected dimension were optimised using a semi-supervised criterion rather than assuming a threshold placement at zero (for mean centered data) was optimal. My semi-supervised objective function combined two valuable signals in a complementary manner: neighbourhood information arising from the input feature space as encoded by a data-point adjacency matrix and neighbourhood information captured by the low-dimensional projection function itself in the form of a projected dimension. I argued that the maximisation of this semi-supervised objective was computationally intractable to achieve in a brute-force manner due to the large search space of possible thresholds. This motivated the exploration of two non-deterministic stochastic search methods, evolutionary algorithms

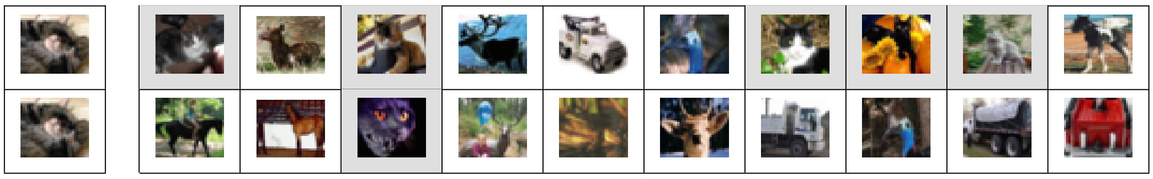


Table 4.20: **Left-most column:** Cat query image. **Top row:** ITQ+NPQ top 10 retrieved images, precision: 0.5. **Bottom row:** ITQ+SBQ top 10 retrieved images, precision: 0.1. Shaded cells indicate true positives.

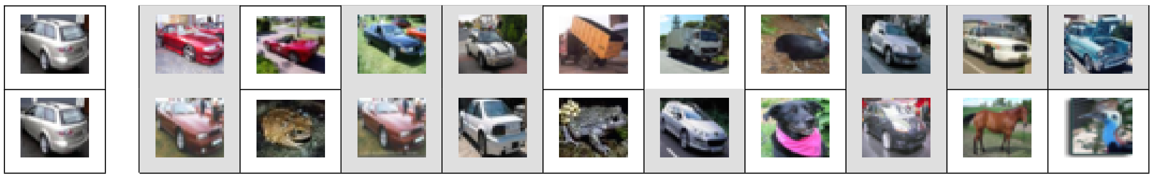


Table 4.21: **Left-most column:** Car query image. **Top row:** ITQ+NPQ top 10 retrieved images, precision: 0.7. **Bottom row:** ITQ+SBQ top 10 retrieved images, precision: 0.5.

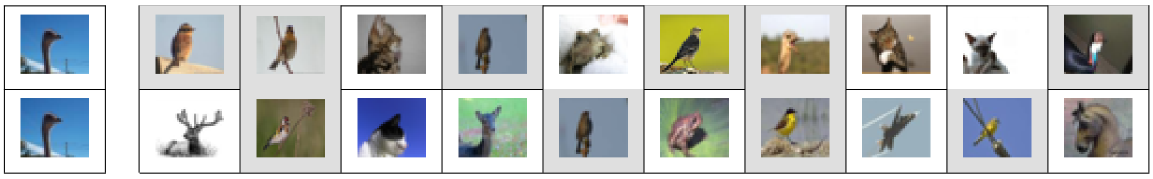


Table 4.22: **Left-most column:** Bird query image. **Top row:** ITQ+NPQ top 10 retrieved images, precision: 0.6. **Bottom row:** ITQ+SBQ top 10 retrieved images, precision: 0.4.

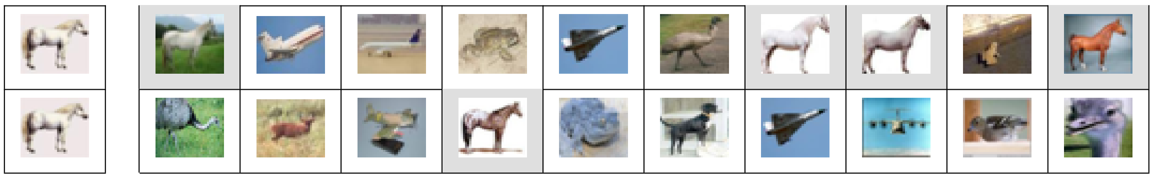


Table 4.23: **Left-most column:** Horse query image. **Top row:** ITQ+NPQ top 10 retrieved images, precision: 0.4. **Bottom row:** ITQ+SBQ top 10 retrieved images, precision: 0.1.

and simulated annealing, both tailored for the threshold optimisation task. In my experimental evaluation relaxing previously ingrained assumptions (single unoptimised threshold per projected dimension) proved to be critical for improving the quality of the binary hashcodes and therefore of the retrieval effectiveness resulting from using those hashcodes for nearest neighbour search. To the best of my knowledge the quantisation model introduced in this chapter is the first scalar quantisation algorithm for hashing-based ANN search that introduces a scheme for semi-supervised threshold placement.

The main experimental findings in this chapter were nine-fold and are summarised hereunder:

- The literature standard method of splitting a dataset (Chapter 3, Section 3.5) into training/testing/validation splits led to a near identical retrieval effectiveness to the improved dataset splitting strategy for all considered baselines and datasets. The results supporting this claim can be found in Section 4.3.3.5 and Tables 4.8-4.9.

- Optimising the position of one or more thresholds per projected dimension *always* yields a higher retrieval effectiveness than the equivalent number of statically placed threshold(s). This claim is validated by Tables 4.8-4.19 in Sections 4.3.3.5-4.3.3.8.

- Quantising LSH projections with a single optimised threshold per projected dimension generally gives a higher retrieval effectiveness than using two or more thresholds. Quantitative results relating to this claim can by found in Sections 4.3.3.5-4.3.3.6 and Tables 4.8-4.11.

- PCA projections always give a higher retrieval effectiveness when multiple (two or more) thresholds are allocated per projected dimension compared to a single threshold. Supporting results can be found in Section 4.3.3.6 and Table 4.12.

- For those projections, such as PCA, that benefit from multiple thresholds I found the Manhattan Hashing Quantisation (MHQ) codebook (natural binary code) and Manhattan distance ranking strategy for hashcode comparison to be more effective than the Double Bit Quantisation (DBQ) codebook and the vanilla binary (0/1) codebook with Hamming distance. Supporting results can be found in Section 4.3.3.8 and Tables 4.17-4.19.

- My multiple threshold quantisation algorithm that positions thresholds by optimising the semi-supervised objective of Equation 4.6 always yields the highest retrieval effectiveness (as measured by area-under-the-precision-recall curve) out of all considered baseline quantisation algorithms and for a wide selection of popular data-dependent projection functions. Leveraging a supervisory signal for threshold placement is critical for maximising retrieval effectiveness. Supporting results can be found in Section 4.3.3.8 and Tables 4.17-4.19.

- Iterative Quantisation (ITQ) significantly outperformed the competing data-depen dent (unsupervised) projection functions of Spectral Hashing (SH) and Principal Components Analysis Hashing (PCAH) for all three datasets and across all considered hashcode lengths. Supporting results can be found in Section 4.3.3.8 and Tables 4.17-4.19.

- The training time of my multi-threshold quantisation algorithm is an order of magnitude faster than the HQ algorithm of Liu et al. (2011) and commensurate with the k-means based MHQ algorithm of Kong et al. (2012). NPQ is therefore significantly more effective than the state-of-the-art quantisation model (MHQ), while maintaining a training time that is also indistinguishable from MHQ. This claim is validated by the results in Section 4.3.3.4 and Table 4.6.

- Evolutionary algorithms provide a more effective method of stochastic search for threshold optimisation than simulated annealing when learning more than a single threshold per projected dimension ($T > 1$). This result is demonstrated in Section 4.3.3.3.

For the Computer Vision practitioner who is perhaps most interested in maximising image retrieval effectiveness in his or her end-application, this chapter could be summarised with the recommendation to use the Iterative Quantisation (ITQ) projection function (Gong and Lazebnik (2011)) coupled with the proposed multi-threshold quantisation model (NPQ) with a setting of $T = 3$ thresholds per projected dimension (Moran et al. (2013a)). This configuration substantially outperformed the other projection function/quantisation model combinations I considered in my experiments. I further note that my quantisation algorithm is not limited to improving the accuracy of nearest neighbour search. As touched upon briefly in Chapter 2, Section 2.5.5 NPQ could be used, for example, in discretising attributes for use in general machine learning models such as Naïve Bayes. This particular investigation, however, is left for future research.

Despite the substantial gains in retrieval effectiveness observed across the board I did notice some potential limitations of the introduced quantisation algorithm that motivates further research. A particularly limiting assumption was that the same allocation of thresholds should be assigned to all projected dimensions for the same projection function, and furthermore that this threshold quantity (denoted by T) should be determined a-priori by the user. In my experimental evaluation I found that different projection functions preferred different allocations of thresholds: from a single threshold per projected dimension for LSH to multiple (2+) thresholds per projected dimension for PCA projections. This finding motivates further exploration in Chapter 5 of novel data-driven schemes for automatically discovering the *optimal number of thresholds* for each projected dimension for a particular projection function, rather than assuming (as I did in this chapter) that the allocation should be uniform and identical for all projected dimensions.

Chapter 5

Learning Variable Quantisation Thresholds

The research presented in this Chapter has been previously published in Moran et al. (2013b).

5.1 Introduction

In Chapter 4, I introduced a new multi-threshold quantisation algorithm for hashing-based approximate nearest neighbour (ANN) search. This semi-supervised quantisation model optimised the position of one or more thresholds along each projected dimension by fusing affinity information on data-point pairs arising from both the high dimensional input feature space and the lower-dimensional projected feature space. The intuitive objective of the algorithm was to position the quantisation thresholds so that the projections for related data-points end up in the same thresholded regions and the projections of dissimilar data-points fall within different regions. Each thresholded region was associated with a unique bitcode from a single or multi-bit binary codebook. To quantise the projection of a data-point I compared the projected value to the quantisation thresholds to pinpoint the appropriate thresholded region and assigned the corresponding bitcode of that region to the data-point. By repeating this procedure for the remaining projected dimensions I was able to build up a K-bit binary hashcode for each of the data-points. The experimental analysis demonstrated that a significant increase in retrieval effectiveness could be achieved from both optimising the positioning of the thresholds to preserve as many must-link and cannot-link relationships between the data-points as possible, and additionally for certain projection functions (such as

PCA), allocating more than a single threshold per projected dimension. The experimental results also suggested that the optimal threshold allocation ($T \in \mathbb{Z}_+$) *varied* according to the specific hash function responsible for generating the low-dimensional projections. For example, LSH was generally shown to prefer a single threshold allocation ($T = 1$) to each projected dimension whereas PCA projections benefited significantly from a multiple-threshold allocation in which, for example, three thresholds ($T = 3$) were allocated to each dimension.

In the previous chapter the quantity of thresholds T allocated per projected dimension was decided *a-priori* and remained the same for all K projected dimensions arising from a specific projection function. In this chapter I expand upon the finding that the threshold allocation is projection function specific by relaxing assumption A_2 outlined in Chapter 1. To this end I argue that the allocation of thresholds should *vary per projected dimension*, rather than being kept uniform across projected dimensions. I further contend that the optimal variable allocation can effectively be found by computing a measure of the *neighbourhood preserving quality* of each projection and assigning more thresholds to those projected dimensions that better conserve the pairwise relationship between data-points in the low-dimensional projected space. In other words I argue that retrieval effectiveness can be further increased by learning a variable allocation of thresholds $T_k \in \mathbb{Z}_+$ for each projected dimension $\mathbf{y}^k \in \mathbb{R}^N$ where the allocation is informed by the quality of the projection. In this chapter I keep with the general theme of this thesis and advocate a data-driven approach to learning a variable threshold allocation subject to a specified threshold budget. To the best of my knowledge the research described in this chapter was the first to introduce and formulate the variable quantisation threshold learning problem in the context of hashing-based ANN search (Moran et al. (2013b)).

The remainder of this chapter is structured as follows: in Section 5.2 I introduce two data-driven algorithms that learn a variable allocation of quantisation thresholds across projected dimensions for a specific projection function. In Section 5.3 I evaluate the two variable threshold quantisation models on their ability to generate effective binary hashcodes for image retrieval. Finally, in Section 5.4 I summarise the main contributions of this chapter and present conclusions arising from the experimental evaluation.

5.2 Variable Quantisation Threshold Allocation

5.2.1 Problem Definition

The problem definition in this chapter is broadly similar to the definition given in the previous chapter (Chapter 4, Section 4.2.1) but with the additional requirement to learn an appropriate number of thresholds for each projected dimension. Concretely the objective in this chapter is to *learn* a set of thresholds $\mathbf{t}_k = [t_{k1}, t_{k2}, \ldots, t_{kT_k}]$ where $t_{ki} \in \mathbb{R}$ and $t_{k1} < t_{k2} \ldots < t_{kT_k}$ for each of the K projected dimensions $\{\mathbf{y}^k \in \mathbb{R}^N\}_{k=1}^K$ while also finding the optimal allocation of thresholds $\{T_k \in \mathbb{Z}_+\}_{k=1}^K$ to each of the K projected dimensions, subject to a total threshold budget $\sum_{k=1}^K \log_2(T_k+1) = K$ and $T_k \in \{0,1,3,7,15\}$. A threshold budget, or equivalently a bit budget, is required because we wish to extract the maximum retrieval effectiveness from as short a hashcode as possible. Short hashcodes ($<$ 128 bits) are particularly useful because they save both time and memory when retrieving nearest neighbours, as compared to much longer hashcodes ($\gg$ 128 bits) or the original high-dimensional feature vectors. In a similar manner to Chapter 4 the quality of the resulting quantisation will be judged by applying the computed hashcodes to the task of query-by-example image retrieval. This threshold allocation problem is computationally difficult given the upper bound of T_{max}^K possible allocations of thresholds, where $T_{max} \in \mathbb{Z}_+$ is the maximum number of thesholds that can be allocated to any given projected dimension. This space of threshold allocations is impossible to search exhaustively therefore necessitating the introduction of more efficient search algorithms to solve the variable threshold allocation problem. It is the specification and evaluation of these algorithms which forms the focus of Section 5.2.2.

5.2.2 Algorithms for Variable Threshold Allocation

In this section I introduce two novel algorithms for learning an allocation of thresholds for each projected dimension based on a numerical measure of the quality of a projected dimension. In Section 5.2.2.1, I describe how this novel quality measure is based on counting the number of pairwise constraints between data-points that are conserved in a projected dimension thresholded with T_k thresholds. This quality measure is the F_β-measure computed on the data-point adjacency matrix $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ and constitutes a minor adaptation of my multi-threshold quantisation objective function first introduced in Chapter 4. Having defined and motivated this measure of projected dimension quality I then introduce two algorithms, dubbed Variable Bit Quantisation (VBQ), for efficiently solving the combinatorial search problem of finding the optimal threshold allocation across projected dimensions. The first algorithm I propose is framed as a Binary Integer Linear Program (BILP) (Section 5.2.2.3) that seeks to allocate thresholds in such a way so as to maximise the cumulative F_β-measure across all projected dimensions subject to an upper bound on the total permissible number of thresholds that can be allocated. The BILP is solved using standard branch-and-bound search. My alternative threshold allocation algorithm is a greedy approach that reallocates quantisation thresholds (Section 5.2.2.4) from lower quality (i.e. low F_β-measure) projected dimensions to projected dimensions that are deemed to be of a higher quality (i.e. higher F_β-measure). This greedy algorithm has a similar effect to the branch-and-bound solution in seeking a threshold allocation which maximises the cumulative F_β-measure. I compare the effectiveness and efficiency of both threshold allocation algorithms in my experimental evaluation in Section 5.3.

5.2.2.1 Judging Projection Quality: F_β-Measure Scoring Function

In this section I argue that counting the number of true nearest neighbours that fall within the same thresholded regions (true positives, TPs), the number of non-nearest neighbours that fall within the same regions (false positives, FPs) and the number of true nearest neighbours that fall within different thresholded regions (false negatives, FNs) is an effective measure of projected dimension quality[1]. In a similar manner to the multi-threshold quantisation algorithm introduced in Chapter 4, the TPs, FPs and FNs are derived from data-point adjacency graph $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$, with $S_{ij} = 1$ if data-points $\mathbf{x}_i, \mathbf{x}_j$ are true nearest neighbours, and $S_{ij} = 0$ otherwise. By quality I refer to the *locality preserving* ability of a projected dimension which is simply the ability to faithfully preserve the neighbourhood relationship between the data-points. For example, if two data-points are close by in the original feature space then their projections should ideally remain within close proximity along the projected dimension, and vice-versa for more distant data-points. If the projections of nearest neighbours are close by then they are more likely to fall within the same thresholded region of the projected dimension and therefore receive the same bit(s). I contend that this hypothetical projected dimension should be deemed to be of high quality as the bits generated from

[1]An unsupervised measure such as variance would also be a possible candidate for measuring the quality of a projected dimension. My threshold allocation algorithms presented in Sections 5.2.2.3-5.2.2.4 could also conceivably be used with variance as the allocation signal. I leave investigation of variance to future work.

the dimension are likely to be similar for many nearest neighbours and dissimilar for non-nearest neighbours, exactly the criteria for building effective hashcodes for image retrieval.

I propose in this section to combine the TPs, FPs and FN counts using the well-known F_β-measure metric from the field of Information Retrieval (IR). I previously touched on the F_β-measure in Chapter 3, Section 3.6.1 in the context of evaluating unranked sets of retrieved images. The F_β-measure is the weighted harmonic mean of recall (R) and precision (P) (Rijsbergen (1979)) which I present again in Equation 5.1 for reading convenience

$$\begin{aligned} F_\beta &= \frac{(1+\beta^2)PR}{\beta^2 P + R} \\ &= \frac{(1+\beta^2)TP}{(1+\beta^2)TP + \beta^2 FN + FP} \end{aligned} \tag{5.1}$$

The parameter $\beta \in \mathbb{R}_+$ specifies the contribution from the precision and recall. Setting $\beta < 1$ in Equation 5.1 weights precision higher than recall, and vice-versa for for a setting of $\beta > 1$. I find in my experimental evaluation in Section 5.3 that it is important to correctly set β when computing a threshold allocation from the F_β-measure scores. The TPs, FPs and FNs for Equation 5.1 are computed in an identical fashion to Chapter 4. To recapitulate, I first build an indicator matrix $\mathbf{P}^k \in \{0,1\}^{N_{trd} \times N_{trd}}$ for projected dimension $\mathbf{y}^k \in \mathbb{R}^N$ that specifies the data-point pairs $(\mathbf{x}_i, \mathbf{x}_j)$ that fall within the same thresholded regions of the projected dimension $\mathbf{y}^k \in \mathbb{R}^N$ (Equation 5.2).

$$P^k_{ij} = \begin{cases} 1, & \text{if} \quad \exists_\gamma \quad s.t. \quad t_{k\gamma} \le (y^k_i, y^k_j) < t_{k(\gamma+1)} \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

Recall that $[t_{k1} \dots t_{kT}]$ denotes the T thresholds partitioning the k^{th} projected dimension. The *true positives* (TP), *false negatives* (FN) and *false positives* (FP) are computed using Equations 5.3-5.5.

$$TP = \frac{1}{2}\sum_{ij} P_{ij} S_{ij} = \frac{1}{2}\|\mathbf{P} \circ \mathbf{S}\|_1 \tag{5.3}$$

$$FN = \frac{1}{2}\sum_{ij} S_{ij} - TP = \frac{1}{2}\|\mathbf{S}\|_1 - TP \tag{5.4}$$

$$FP = \frac{1}{2}\sum_{ij} P_{ij} - TP = \frac{1}{2}\|\mathbf{P}\|_1 - TP \tag{5.5}$$

With these definitions the F_β-measure for a particular thresholding of an arbitrary projected dimension $\mathbf{y}^k$ can be specified in matricial form as given in Equation 5.6

$$F_\beta(\mathbf{t}_k) = \frac{(1+\beta^2)\|\mathbf{P}\circ\mathbf{S}\|_1}{\beta^2\|\mathbf{S}\|_1 + \|\mathbf{P}\|_1} \tag{5.6}$$

In Chapter 4, Section 4.3.3.2 I found that it is optimal to set the interpolation parameter between the supervised (F_β-measure) and unsupervised terms to $\alpha = 1$, for hashcode lengths up to 128 bits, when using the Manhattan hashing quantisation codebook. In this chapter I therefore do not interpolate the F_β-measure with an unsupervised term as I did in Equation 4.6 in Chapter 4. Investigating the benefit, or otherwise, of interpolation with an unsupervised signal in the context of variable threshold allocation is left to future work.

In this chapter my objective is to find both, the optimal number of thresholds to allocate to each projected dimension, and the optimal placement of those thresholds. The possible quantity of thresholds for each dimension T_k is constrained by the Manhattan quantisation codebook to be within the finite set $T_k \in [0,1,3,7,15]$. I argue that projected dimensions with a higher locality preserving quality should be given a larger allocation of the available thresholds so that the neighbourhood structure captured by that dimension can be maximally exploited. To grade the quality of a threshold allocation I find the optimal position of the thresholds by maximising Equation 5.6 using evolutionary algorithms (EA). The resulting F_β-measure, relating to the optimal position of the T_k thresholds as found by the EA, is then used as the indicator of projected dimension quality. Importantly once the optimal threshold positions are computed they do not have to be re-optimised in order to perform the threshold allocation. This offline training procedure will yield five F_β-measure scores per projected dimension, one for each threshold quantity $T_k \in [0,1,3,7,15]$.

The F_β-measure scores are then used by my variable threshold allocation algorithm to compute the threshold allocation that yields the highest *cumulative* F_β-measure subject to a fixed threshold allocation budget. In doing my contention is that the F_β-measure varies widely for each possible threshold quantity $T_k \in [0,1,3,7,15]$ and that there is a unique quantity of thresholds that provides an overall greatest F_β-measure for a given projected dimension. In addition I argue that the value of the maximum F_β-measure is higher for projected dimensions that are more effective at preserving the pairwise constraints between data-points in the adjacency matrix $\mathbf{S}$. If this hypothesis is correct then the optimised F_β-measure will most likely provide an effective signal for threshold allocation in a way that assigns a greater proportion of the thresholds to those projected dimensions that respect the locality structure encoded in the adjacency graph $\mathbf{S}$.

Figure 5.1 provides a further intuition as to why assigning a projected dimension a threshold quantity T_k that maximises Equation 5.6 might be expected to lead to an effective multi-threshold quantisation. In this toy example I show a two-dimensional (2D) input feature space in which data-points are represented by coloured shapes. Those data-points with the same shape and colour are nearest neighbours in the 2D plane. Two hyperplanes $\mathbf{h}_1 \in \mathbb{R}^2, \mathbf{h}_2 \in \mathbb{R}^2$ are shown partitioning the space into four regions. The hyperplanes $\mathbf{h}_1 \in \mathbb{R}^2, \mathbf{h}_2 \in \mathbb{R}^2$ have normal vectors $\mathbf{w}_1 \in \mathbb{R}^2, \mathbf{w}_2 \in \mathbb{R}^2$, respectively. On the bottom diagram in Figure 5.1, I show the resulting projected dimensions $\mathbf{y}^1 \in \mathbb{R}^2, \mathbf{y}^2 \in \mathbb{R}^2$ obtained by projecting the data-points onto the normal vectors. The value of Equation 5.6 for projected dimension $\mathbf{y}^2$ with $\beta = 1$ is at the maximum value ($F_1 = 1$). Observing the quantised regions we see that all nearest neighbours are located beside each other and are not partitioned by any threshold. This can be deemed a perfect quantisation of the projected dimension because all nearest neighbours will be assigned the same hashcode. This fact is highlighted by the high value of Equation 5.6. Furthermore we observe that three thresholds is the minimal quantity of thresholds in this contrived example that will lead to a perfect quantisation, with either more ($T > 3$) or less thresholds ($T < 3$) receiving a lower F_β-measure score. The opposite is the case for projected dimension $\mathbf{y}^1$. Given the high mixing of the data-points *no* thresholds do just as well in this situation as one or more thresholds. The inability of this hyperplane to clump together related data-points along the projected dimension is highlighted by a low F_β-measure. In this case the corresponding hyperplane ($\mathbf{h}_1$) should ideally be pruned and the allocation of thresholds distributed to the more effective hyperplane ($\mathbf{h}_2$).

5.2.2.2 A Link to the Laplacian Score

My application of the F_β-measure to allocate thresholds to projected dimensions is in some senses acting as a filter-based feature selection score that has been constructed specifically for hashing-based ANN search in which multiple thresholds are used for

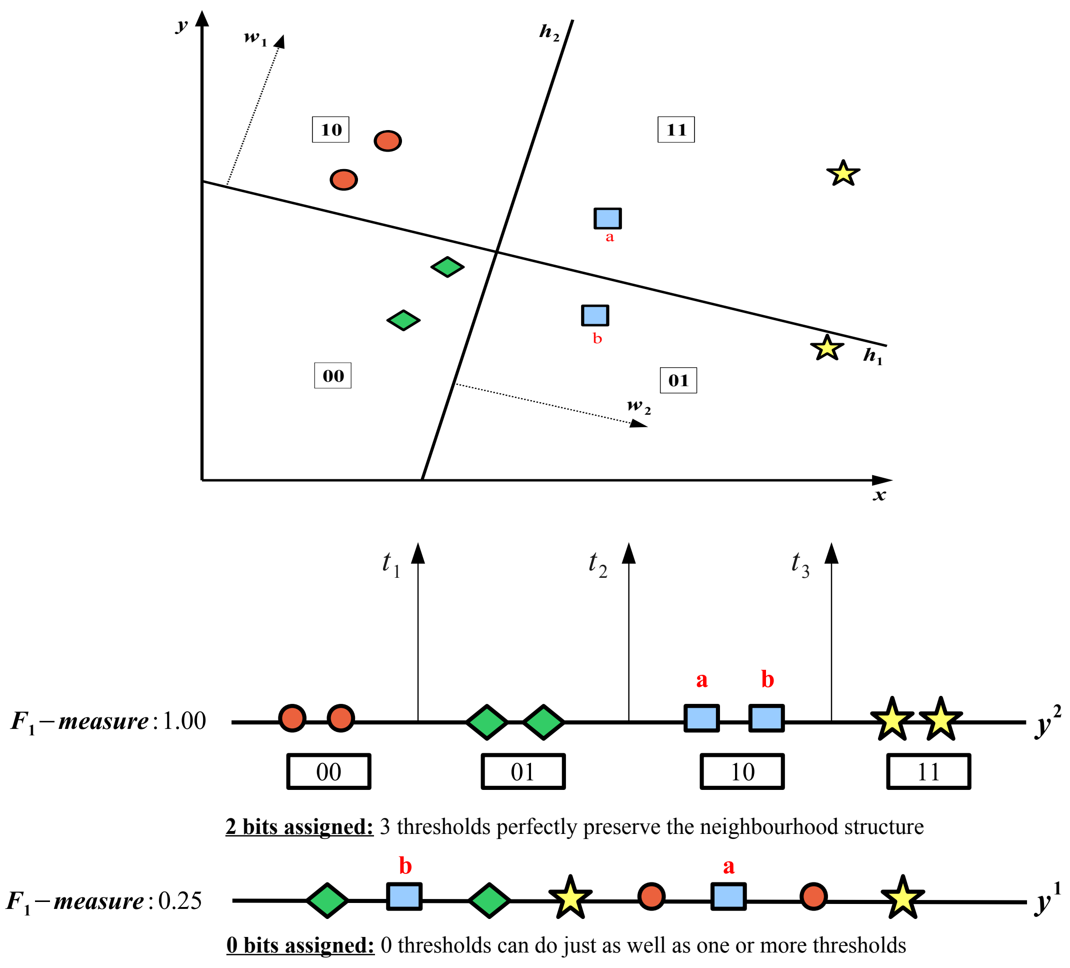

y
x
w_1
w_2
h_1
h_2
10
11
00
01
a
b
t_1
t_2
t_3
$F_1-measure: 1.00$
y^2
00
01
10
11
2 bits assigned: 3 thresholds perfectly preserve the neighbourhood structure
$F_1-measure: 0.25$
y^1
0 bits assigned: 0 thresholds can do just as well as one or more thresholds

Figure 5.1: Intuition behind the application of the F_β-measure as a means of grading the quality of a quantisation resulting from an assignment of a given number of thresholds. I set $\beta = 1$ for the purposes of this example. The top diagram illustrates a 2D plane in which data-points (indicated by the coloured shapes) are embedded. The diagram below illustrates the projection of the data-points onto the normal vectors $\mathbf{w}_1, \mathbf{w}_2$. Projected dimension $\mathbf{y}^2$ can be perfectly quantised with 3 thresholds. In the case of data-points a, b both end up within the same region. This is not the case for projected dimension $\mathbf{y}^1$ which is much more difficult to quantise due to related data-points ending up much farther away from each other along the dimension. The reader is guided to Section 5.2.2.1 for a further and fuller description.

quantisation. By viewing my algorithm under the lens of feature selection I can draw an interesting parallel to the Laplacian score (He et al. (2005)), a popular supervised feature selection score which originated in the field of Machine Learning. The Laplacian score is a filter-based feature selection metric which can be applied independent of a classifier, the latter bond being a requirement of wrapper-based algorithms. Briefly

Laplacian Score is reminiscent of the Laplacian Eigenmap which I discussed in detail in Chapter 2, Section 2.6.4.4 in the context of the Self Taught Hashing (STH) model. The Laplacian Eigenmap seeks a low dimensional projection of a set of data-points in a way that preserves the pairwise relationships between those data-points as encoded in the adjacency graph $\mathbf{S}$. Data-points that are neighbours according to the adjacency graph ($S_{ij} = 1$) should end up close together when projected into the low-dimensional space. Laplacian Score further develops this key idea into a feature selection metric that assigns lower (lower is better) scores to features that are related ($S_{ij} = 1$) and which are also closer together along the projected dimension, that is the distance $(y_i^k - y_j^k)^2$ between the projections of data-points $\mathbf{x}_i, \mathbf{x}_j$ is low (Equation 5.7)

$$L_k = \sum_{ij} \frac{(y_i^k - y_j^k)^2 S_{ij}}{var(\mathbf{y}^k)} \tag{5.7}$$

where L_k is the Laplacian score of the k^{th} dimension and $var(\mathbf{y}^k)$ computes the variance of the dimension $\mathbf{y}^k \in \mathbb{R}^N$, that is $var(\mathbf{y}^k) = 1/N_{trd} \sum_{i=1}^{N_{trd}} (y_i^k - \mu)^2$, and μ denotes the mean of the projected dimension. The Laplacian score is minimised for features that respect the graph structure, while also having a large variance and therefore presumably high representational power. In a different manner to the Laplacian Score, my application of the F_β-measure explicitly takes into account how many true pairs end up within the same thresholded regions but does not account for the closeness of their corresponding projections.

5.2.2.3 Threshold Allocation via Branch-and-Bound

Having defined the metric I use to grade the quality of a projected dimension in Section 5.2.2.1 I will now introduce my first algorithm that leverages the resulting quality scores to learn an effective distribution of thresholds across K projected dimensions. As I argued in Section 5.2.2.1 the F_β-measure scores per hyperplane ($\mathbf{h}_k$), per threshold count ($T_k \in [0, 1, 3, 7, 15]$) are an effective signal for threshold allocation as more informative hyperplanes tend to have higher F_β-measures for a higher number of thresholds. I hypothesise that seeking the threshold allocation that maximises the total F_β-measure as accumulated across all K projected dimensions, subject to a threshold budget, is a suitable objective for optimisation. Unfortunately this combinatorial optimisation problem is NP-hard which can be immediately deduced with analogy to the binary *knapsack problem*, a classic problem in combinatorial optimisation (Dantzig (1957)). The binary knapsack problem involves selecting a number of items to place into a knapsack that maximises the total value of the items while adhering to a limit on the total weight. An item can only be chosen for inclusion once, hence the binary nature of the problem (0 is exclude from knapsack, 1 is include in knapsack). The inherent difficulty of this problem stems from this integrality constraint. This is exactly my threshold allocation problem in which an "item" is one or more thresholds for a dimension, the "weight" is the threshold quantity for that dimension and the value is the F_β-measure for that number of thresholds when positioned optimally along the projected dimension. By drawing a parallel to the task in this way I can benefit from the rich literature that has already been established on approximately solving the binary knapsack problem (Martello and Toth (1990)). My first solution has indeed been inspired in this manner and involves framing the learning objective as a *binary integer linear program* (BILP).

To construct a BILP appropriate for my purposes I firstly collate the F_β-measure scores per hyperplane, per threshold count in a matrix $\mathbf{F} \in \mathbb{R}^{(B_{max}+1)\times K}$ with elements F_{bk} which represent the accuracy that results from allocating $2^b - 1$ thresholds to dimension k. In this case $b \in \{0,\dots,B_{max}\}$ indexes the rows, with B_{max} being the maximum number of bits allowable for any given hyperplane, and $k \in \{1\dots,K\}$ indexes the columns of the F_β-measure matrix. Note the equivalence between thresholds and bits: if we have T_k thresholds for a particular projected dimension this equates to an allocation of $B_k = \log_2(T+1)$ bits for that dimension. The BILP uses $\mathbf{F}$ to find the threshold allocation that maximises the cumulative F_β-measure across the K hyperplanes (Equation 5.8)

$$\begin{aligned}
\max \quad & ||\mathbf{F}\circ\mathbf{Z}|| \\
\text{subject to } & ||\mathbf{Z}_c|| = 1 \qquad c \in \{1\dots K\} \\
& ||\mathbf{Z}\circ\mathbf{D}|| \leq K \\
& \mathbf{Z} \quad \text{is binary}
\end{aligned} \tag{5.8}$$

where $||.||$ denotes the Frobenius L_1 norm, $\circ$ the Hadamard (elementwise) product and $\mathbf{D} \in \mathbb{Z}_+^{(B_{max}+1)\times K}$ is a constraint matrix, with $D_{bh} = b - 1$, ensuring that the threshold allocation remains within the bit budget B. I now give an intuitive overview of each term in Equation 5.8:

- $||\mathbf{F}\circ\mathbf{Z}||$: This term computes the cumulative F_β-measure score for bit allocation specified by $\mathbf{Z}$

- $\|\mathbf{Z}_c\| = 1$: This constraint ensures that a specific number of bits selected from the set $b \in [0,1,2,3,4]$ is allocated to each hyperplane, with $B_{max} = 4$. This constraint also ensures that more than one value from this set cannot be allocated to any single hyperplane.

- $\|\mathbf{Z} \circ \mathbf{D}\|$: This constraint enforces the bit allocation $\mathbf{Z}$ to be less than or equal to the available bit budget K.

The BILP is solved using the off-the-shelf *branch and bound* optimisation algorithm (Land and Doig (1960))[2]. The branch and bound algorithm is a common workhorse for solving integer programming problems. Branch and bound enumerates the entire space of candidate solutions but maintains tractability by discarding large portions of the search space based on estimated lower and upper bounds on the quantity being optimised. Describing the specifics of this solver is beyond the scope of this thesis, however the reader is pointed to Brassard and Bratley (1996) for an introductory overview. The output from the branch and bound BILP solver is an indicator matrix $\mathbf{Z} \in \{0,1\}^{(B_{max}+1)\times K}$ whose columns specify the optimal threshold allocation for a given hyperplane, that is, $Z_{bk} = 1$ if the BILP decided to allocate b bits (equivalently $2^b - 1$ thresholds) for the k^{th} hyperplane, and zero otherwise. I compute the quality score of a zero bit allocation by computing the F_β-measure on a projected dimension with an allocation of zero thresholds. Example input and output from the branch and bound algorithm for the toy problem in Figure 5.1 are given in matrices $\mathbf{F}, \mathbf{D}, \mathbf{Z}$ below (in this example, $B_{max} = 2$ and $K = 2$). The indicator matrix $\mathbf{Z}$ is output by the branch and bound solver as the best feasible solution found subject to the problem constraints.

$$
\begin{array}{cc}
\mathbf{F} & \begin{array}{cc} k_1 & k_2 \end{array} \\
\begin{array}{c} b_0 \\ b_1 \\ b_2 \end{array} & \begin{pmatrix} 0.25 & 0.25 \\ 0.35 & 0.50 \\ 0.40 & 1.00 \end{pmatrix}
\end{array}
\quad
\begin{array}{c}
\mathbf{D} \\
\begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 2 & 2 \end{pmatrix}
\end{array}
\quad
\begin{array}{c}
\mathbf{Z} \\
\begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}
\end{array}
$$

Notice how the indicator matrix $\mathbf{Z}$ specifies an assignment of 0 bits (0 thresholds) for hyperplane $\mathbf{h}_1$ and 2 bits (3 thresholds) for hyperplane $\mathbf{h}_2$ as this yields the highest cumulative F_1-measure (1.25) across the K projected dimensions while also meeting the required threshold budget. This result accords with our intuition in how the thresholds should be allocated in the toy example of Figure 5.1. The BILP as framed in

[2]Specifically I use the `bintprog` Matlab solver with default parameters.

Equation 5.8 is therefore a principled method for both selecting a discriminative subset and allocating an appropriate quantity of thresholds to those hyperplanes subject to a fixed overall threshold budget. The computational time complexity of solving the BILP using branch and bound has exponential $(O(2^{(B_{max}+1)K})$ time complexity in the worst-case (Lawler and Wood (1966)), although in practice the tree search is actually very efficient. In general I find that the branch and bound cost is negligible in comparison to the dominant training time cost of computing the $(B_{max}+1)K$ F_β-measure scores in the matrix $\mathbf{F}$ which is $O(T_{max}N_{trd}^2F + KN_{trd}^2)$, with N_{trd} denoting the number of training data-points, F the number of objective function evaluations (Equation 5.6) and $T_{max} = K\sum_{b=1}^{B_{max}} 2^b - 1$. I will now introduce an alternative bit allocation algorithm that improves upon the computational time complexity of the branch-and-bound solution by eliminating the dependence on B_{max}.

5.2.2.4 Greedy Threshold Allocation Algorithm

The branch-and-bound solution presented in Section 5.2.2.3 is dominated by the computation of the $(B_{max}+1)K$ F_β-measure scores taking $O(T_{max}N_{trd}^2F + KN_{trd}^2)$ time. In my second contribution of this chapter I introduce a novel greedy algorithm for threshold allocation that reduces this computational time complexity to $O(T_{gdy}N_{trd}^2F + KN_{trd}^2)$, where $T_{gdy} \ll T_{max}$ and is independent of B_{max}. This algorithm performs the same task as the BILP but is greedy and hence it is expected to be faster and simpler. This algorithm is based on a simple and intuitive idea, namely the redistribution of thresholds at each iteration from lower quality (low F_β-measure scores) hyperplanes to higher quality (higher F_β-measure scores) hyperplanes while adhering to the specified threshold budget. I use a new notation in this section to indicate the number of bits assigned per projected dimension. Rather than use an indicator matrix $\mathbf{Z}$ as in Section 5.2.2.3, I unroll $\mathbf{Z}$ as a vector of bit counts $\mathbf{z} \in \mathbb{Z}_+^K$, which will make the exposition of the algorithm somewhat easier. For example $z_h = 3$ if we have allocated 3 bits ($2^3 - 1 = 7$ thresholds) to the h^{th} hyperplane. I initialise $z_h = 1, \forall_h$ so at the start of the algorithm we have allocated one threshold per hyperplane. The matrix of F_β-measures $\mathbf{F} \in \mathbb{R}^{(B_{max}+1)\times K}$ is computed in the same manner as for the binary integer linear program (BILP) based solution outlined in Section 5.2.2.3. To recapitulate, to compute F_{bh} I position $2^b - 1$ thresholds along the h^{th} projected dimension to maximise Equation 5.6. The quantisation threshold positioning is found using evolutionary algorithms as outlined in Chapter 4, Section 4.2.3.2. Element F_{bh} of matrix $\mathbf{F}$ is then set to the maximum F_β-measure arising from the optimal threshold configuration discovered by the stochastic search. Crucially, however, I do not compute all $(B_{max}+1)K$ F_β-measure scores at once as for the branch-and-bound solution. Rather the greedy algorithm permits a more efficient *on-demand* computation of the F_β-measure scores.

I initialise the **F** matrix by computing the F_β-measure score for $B = 0, 1, 2$ bits, requiring $O(3K)$ computations. Having initialised **F**, I learn the optimal bit allocation **z** through *threshold redistribution*. In the first iteration this procedure involves finding a projected dimension that would "most benefit" (largest delta in F_β-measure) from having its bit allocation increased by one unit and the projected dimension that would "suffer least" (smallest delta in F_β-measure) from having a bit subtracted from its bit allocation. To determine the projected dimension that should have a bit added to its allocation I simply search for the projected dimension h_{max} where $h_{max} = \text{argmax}_h(F_{2,h} - F_{1,h})$. Intuitively the projected dimension that has the greatest *increase* in F_β-measure between its F_β-measure for its current allocation of 1 bit and its F_β-measure for a potential allocation of 2 bits, should be assigned that additional 1 bit. Having allocated an additional bit to projected dimension h_{max} we have now allocated $K+1$ bits, 1 bit over the maximum bit quota of K. To respect the quota I remove a bit from the hyperplane which has the *lowest difference* between its F_β-measure at a bit allocation of 1 and its F_β-measure for a bit allocation of 0 bits.

More formally we wish to find hyperplane h_{min} where $h_{min} = \text{argmin}_h(F_{1,h} - F_{0,h})$. In allocating 0 bits to h_{min} we have effectively discarded that hyperplane and eliminated it from further consideration. Note that once a hyperplane has 0 bits or B_{max} bits assigned the count for that hyperplane forever remains fixed at those values. By adding a bit to the hyperplane that contributes the greatest increase in F_β-measure while removing a bit from the hyperplane that loses the least F_β-measure I greedily approximate a maximisation of the cumulative F_β-measure across all K hyperplanes. The algorithm proceeds in this manner until we reach a point where the maximum increase in F_β-measure is lower than the minimum decrease in F_β-measure, which indicates that there is no more benefit from redistributing thresholds amongst the K projected dimensions. On each of these subsequent steps post initialisation of the **F** matrix, the computationally expensive operation of computing the F_β-measure needs only to be computed *once per step* namely for that hyperplane that was recently promoted to have the maximum current number of bits. A pseudocode version of the algorithm is presented in Algorithm 8.

This greedy threshold allocation algorithm has a training time complexity characterised by $O(T_{gdy}N_{trd}^2F + KN_{trd}^2)$ and a threshold allocation time complexity of $O(1)$,

Algorithm 8: Greedy Threshold Allocation Algorithm

```
Input: Projected dimensions $\{\mathbf{y}^k \in \mathbb{R}^N\}_{k=1}^K$, the maximum number of bits per
       dimension $B_{max} \in [1,2,3,4]$, $F_\beta$-measure scores $\mathbf{F} \in \mathbb{R}^{(B_{max}+1)\times K}$
Output: Optimal allocation of thresholds to projected dimensions $\mathbf{z} \in \mathbb{Z}_+^K$
1  Set $F_\beta^{max} = \infty$, $F_\beta^{min} = 0$
2  Set $\mathbf{z} = \mathbf{1}^K$                              // Initalise bit counts to 1
3  while $F_\beta^{max} > F_\beta^{min}$ do
4      $h_{min} = \text{argmin}_h(F_{z_h h} - F_{z_h - 1 h})$
5      $F_\beta^{min} = F_{z_{h_{min}} h_{min}} - F_{z_{h_{min}-1} h_{min}}$
6      $h_{max} = \text{argmax}_h(F_{z_h + 1 h} - F_{z_h h})$
7      $F_\beta^{max} = F_{z_{h_{max}} + 1 h_{max}} - F_{z_{h_{max}} h_{max}}$
8      if $F_\beta^{max} > F_\beta^{min}$ then
9          $z_{h_{max}} = z_{h_{max}} + 1$    // Increment bit count for hyperplane $h_{max}$
10         if $z_{h_{max}} = B_{max}$ then
11             $\mathbf{F}_{\bullet h_{max}} = NaN$             // $h_{max}$ cannot be chosen again
12         end
13         $z_{h_{min}} = z_{h_{min}} - 1$    // Decrement bit count for hyperplane $h_{min}$
14         if $z_{h_{min}} = 0$ then
15             $\mathbf{F}_{\bullet h_{min}} = NaN$             // $h_{min}$ cannot be chosen again
16         end
17     end
18 end
19 return z
```

where $T_{gdy} \ll T_{max}$ and $T_{max} = K\sum_{b=1}^{B_{max}} 2^b - 1$. The training time complexity is independent of B_{max}, and so the greedy threshold learning algorithm is expected to be faster at learning than the branch-and-bound solution presented in Section 5.2.2.3. The reasoning for this is as follows: prior to the execution of the algorithm only the first three rows of matrix $\mathbf{F} \in \mathbb{R}^{(B_{max}+1)\times K}$ need be initialised, relating to a bit allocation of 0, 1 and 2 bits for each of the K available hyperplanes. The total number of F_β-measure computations for this particular initialisation is of $O(3K)$. I contend that only $O(K)$ additional F_β-measure computations are then required for the learning step. The reason for this is that VBQ$_{bound}$ needs only to compute the F_β-measure once per subsequent iteration. This F_β-measure is computed for the hyperplane that was most recently (i.e.

at the previous iteration) promoted to have two or more bits, as this will be the only hyperplane for which the F_β-measure is unknown for an additional ($+1$) bit over its current allocation. Furthermore, the algorithm needs only to perform this computation less than K times as there are only K available bits. The greedy algorithm therefore needs only $O(K)$ F_β-measure computations, independent of the B_{max} term, and we can therefore expect a lower learning time compared to the branch-and-bound algorithm introduced in Section 5.2.2.3. Having just scored the hyperplanes for differing threshold quantities in the learning step, the threshold allocation step is straightforward as we simply use the allocation specified at the end of the final iteration. The threshold allocation time complexity of this algorithm is therefore $O(1)$. A comparison between the learning and allocation runtimes of my proposed greedy and branch-and-bound algorithms are presented in Section 5.3.3.4.

5.3 Experimental Evaluation

5.3.1 Experimental Configuration

In this section I will experimentally test my two variable threshold quantisation algorithms introduced in Section 5.2. The experimental setup will be almost identical to the literature standard configuration used to evaluate my multi-threshold quantisation algorithm in Chapter 4, Section 4.2 (Kong et al. (2012); Kong and Li (2012a,b); Kulis and Darrell (2009); Raginsky and Lazebnik (2009); Gong and Lazebnik (2011)). Specifically my task will be query-by-example image retrieval with the experimental parameters shown in Table 5.1. As a baseline I will compare against my multi-threshold quantisation model (NPQ) introduced in Chapter 4. The NPQ model was shown to significantly outperform competing scalar quantisation models in the literature (MHQ, DBQ, SBQ) that assign a uniform quantity of thresholds across projected dimensions.

I structure the experiments in this chapter to answer the following two main hypotheses:

- H_1: *Allocating a* variable *number of thresholds can yield a higher retrieval effectiveness than a* uniform allocation *of thresholds (NPQ).*
- H_2: *Branch-and-bound (VBQ_{bound}) finds a threshold allocation that leads to a significantly higher retrieval effectiveness than the greedy threshold allocation algorithm (VBQ_{greedy}).*

Parameter	**Setting**	**Chapter Reference**
Groundtruth Definition	ε-NN	Chapter 3, Section 3.3
Evaluation Metric	AUPRC	Chapter 3, Section 3.6.3
Evaluation Paradigm	Hamming Ranking	Chapter 3, Section 3.4
Random Partitions	10	Chapter 3, Section 3.5
Number of Bits (K)	16-128	Chapter 2, Section 2.4

Table 5.1: Configuration of the main experimental parameters for the results presented in this chapter.

Method	**# Thresholds/dim**	**Encoding**	**Ranking Strategy**
VBQ	Variable	NBC	Manhattan distance
NPQ	3	NBC	Manhattan distance

Table 5.2: Parametrisation of the quantisation models studied in this chapter.

Hypothesis H_1 will examine whether a variable threshold allocation can yield a higher retrieval effectiveness than a uniform assignment. Hypothesis H_2 will confirm whether the sub-optimal search strategy of the greedy approach results in an inferior threshold allocation compared to branch-and-bound.

In order to abstract from the effect of the codebook and the hashcode ranking strategy I parametrise all quantisation models in this chapter to use the Manhattan Hashing Quantisation (MHQ) codebook with the Manhattan distance as the hashcode ranking strategy (Table 5.2)[3]. Furthermore to ensure easy replication of my results by interested researchers the experiments in this chapter will be carried out on the standard LabelMe, CIFAR-10 and NUS-WIDE image collections as described in Chapter 3, Section 3.2.1. In a similar manner to my evaluation in Chapter 4, I follow previously accepted procedure in the literature (Kong et al. (2012), Kong and Li (2012a)) and randomly select $N_{teq} = 1{,}000$ data points as testing queries ($\mathbf{X}_{teq} \in \mathbb{R}^{N_{teq} \times D}$), with the remaining points ($\mathbf{X}_{db} \in \mathbb{R}^{N_{db} \times D}$) being used as the database upon which to learn and test the hash functions according to either the literature standard or improved dataset splitting strategy. A further breakdown on the specific dataset splits I use in this chapter are shown in Tables 5.3-5.4.

[3]The reader is guided to Chapter 2, Section 2.5.4 for an overview of this codebook and hashcode ranking strategy.

Partition	**LABELME**	**CIFAR-10**	**NUS-WIDE**
Test queries (N_{teq})	1,000	1,000	1,000
Validation queries (N_{vaq})	1,000	1,000	1,000
Validation database (N_{vad})	10,000	10,000	10,000
Training database (N_{trd})	2,000	2,000	10,000
Test database (N_{ted})	8,000	46,000	247,648

Table 5.3: 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.

Partition	**LABELME**	**CIFAR-10**	**NUS-WIDE**
Test queries (N_{teq})	1,000	1,000	1,000
Validation queries (N_{vaq})	1,000	1,000	1,000
Validation database (N_{vad})	10,000	10,000	10,000
Training database (N_{trd})	2,000	2,000	10,000
Test database (N_{ted})	21,000	59,000	268,648

Table 5.4: 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.

5.3.2 Parameter Optimisation

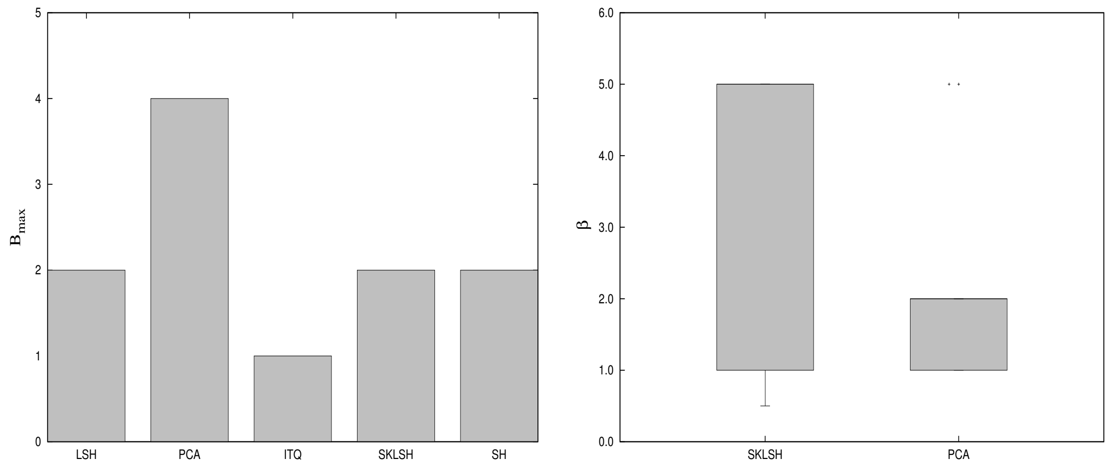
To set the hyperparameters of my model, I conduct a grid search for the best fitting β, B_{max} hyperparameters over the the values $\beta \in \{0.5, 1, 2, 5, 10\}$ and $B_{max} \in \{1, 2, 3, 4\}$. This grid search is performed on the held out validation dataset ($\mathbf{X}_{vaq} \in \mathbb{R}^{N_{vaq} \times D}$, $\mathbf{X}_{vdb} \in \mathbb{R}^{N_{vdb} \times D}$), selecting the parameter combination that maximises validation dataset AUPRC. As the NPQ model introduced in Chapter 4 has a free parameter β for the F_β-measure term I also optimise this parameter for NPQ on the validation portion of the dataset. I refer to the optimised NPQ model as NPQ$_{opt}$ in my result tables. For VBQ, in all experiments the thresholds and threshold allocation are learnt on the training database ($\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D}$). The learnt thresholds are then subsequently used to quantise the testing dataset projections ($\mathbf{X}_{teq} \in \mathbb{R}^{N_{teq} \times D}$, $\mathbf{X}_{tdb} \in \mathbb{R}^{N_{ted} \times D}$). The same procedure is used to learn the thresholds for the NPQ and NPQ$_{opt}$ baselines. The reported AUPRC figures for the test dataset are the average over ten random dataset splits as explained in Chapter 3, Section 3.5. The Wilcoxon signed rank test (Wilcoxon (1945)) is used to determine the statistical significance of the retrieval results. In this case, when comparing system A to system B, a pair of AUPRC values arising from a retrieval run on the current fold forms the unit of the significance test. In all presented result tables, ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over NPQ$_{opt}$, while ▲/▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.05$) over NPQ$_{opt}$.

5.3.3 Experimental Results

5.3.3.1 Experiment I: Effect of the β and B_{max} hyperparameters

The VBQ$_{bound}$ and VBQ$_{greedy}$ models have two hyperparameters that need to be set on a held-out validation dataset: $\beta \in \{0.5, 1, 2, 5, 10\}$ that determines the importance given to precision versus recall in Equation 5.6, and $B_{max} \in \{1, 2, 3, 4\}$ that indicates the maximum number of bits that can be allocated per projected dimension. Figure 5.2 (a) plots the optimal value of B_{max} at 32 bits on CIFAR-10 for five data-independent and dependent projections functions: LSH, PCA, SKLSH, SH and ITQ. The optimal value of the B_{max} parameter varies between each of these different projection functions. Projection functions such as PCA, which produce hyperplanes of widely different neighbourhood preserving quality, benefit a greater maximum number of bits that can be allocated to the small subset of high quality hyperplanes, thereby boosting their contribution to the construction of the hashcodes. ITQ on the other hand attempts to make the quality of the K hyperplanes equal by rotating the input feature space such that the variance captured by each hyperplane is approximately balanced. In this case there is no subset of very high quality hyperplanes that should attract the majority of the available bit budget and therefore the B_{max} parameter in this case is quite intuitively set to $B_{max} = 1$.

Figure 5.2 (b) shows the variation in β for SKLSH and PCA projections across each random dataset split. I observe that β has a larger variance for SKLSH projections with the preferred setting falling between $\beta \in [0.5, 5]$, while for PCA projections β appears more stable with $\beta \in [1, 2]$ generally appearing optimal across most random dataset splits. This latter observation is a consequence of the widely differing partitions induced by the randomly sampled SKLSH hyperplanes compared to the deterministic setting of the PCA hyperplanes across each random dataset split. The variance in β, particularly for SKLSH, suggests that for some partitionings of the input feature space



(a) B_{max} **parameter** (b) β **parameter**

Figure 5.2: Figure (a) shows the optimal value of the B_{max} parameter per projection function. Figure (b) shows the variability of the β parameter per random dataset split for LSH and PCA projections. Results were obtained for the CIFAR-10 dataset at 32 bits.

it is more detrimental to separate pairs of similar images in different quantised regions ($\beta > 1$), compared to placing dissimilar images in the same region, and vice-versa for $\beta < 1$.

5.3.3.2 Experiment II: Variable versus Uniform Threshold Allocation

In this experiment I will study the primary hypothesis (H_1) of this chapter, namely that a variable threshold allocation adapted to specific hyperplanes will yield a higher retrieval effectiveness for nearest neighbour search compared to a uniform allocation of thresholds. To examine this hypothesis I will compare both variable threshold allocation algorithms (VBQ$_{bound}$, VBQ$_{greedy}$) against the strong baseline of NPQ, my own multi-threshold quantisation model (NPQ) introduced in Chapter 4. NPQ assigned a uniform number of thresholds to every projected dimension irrespective of the differing locality preserving qualities of the corresponding hyperplanes. I also explore the effect of tuning the NPQ $\beta \in \{0.5, 1, 2, 5, 10\}$ parameter in this chapter as it could conceivably result in an added boost in retrieval effectiveness. This optimised variant is referred to as NPQ$_{opt}$ in all experiments. Comparing directly to NPQ will therefore tease apart the effect of an informed variable allocation of thresholds. To ensure a meaningful comparison VBQ, NPQ and NPQ$_{opt}$ are constrained to have an identical amount of supervision, namely $N_{trd} = 2,000$ for CIFAR-10 and LabelMe and $N_{trd} = 10,000$ for

	Quantisation Model			
	VBQ$_{bound}$	**VBQ**$_{greedy}$	**NPQ**	**NPQ**$_{opt}$
LSH	0.2035 (0.2105)	**0.2176** (0.2169)▲	0.1621 (0.1662)	0.1742 (0.1744)
ITQ	0.3278 (0.3229)	0.3354 (0.3349)▼▼	0.3917 (0.3898)	**0.3947** (0.3943)
SH	**0.2549** (0.2546)▲▲	0.2299 (0.2284)	0.1834 (0.1871)	0.2044 (0.2038)
PCA	0.2712 (0.2707)	**0.2739** (0.2728)▲▲	0.1660 (0.1683)	0.1833 (0.1834)
SKLSH	0.1830 (0.1827)	**0.1832** (0.1831)▲▲	0.1063 (0.1088)	0.1070 (0.1179)

Table 5.5: AUPRC on the CIFAR-10 dataset with a hashcode length of 32 bits. The quantisation algorithms listed on the first row are used to quantise the projections from the hash functions in the first column. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over NPQ$_{opt}$. ▲/▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.05$) over NPQ$_{opt}$.

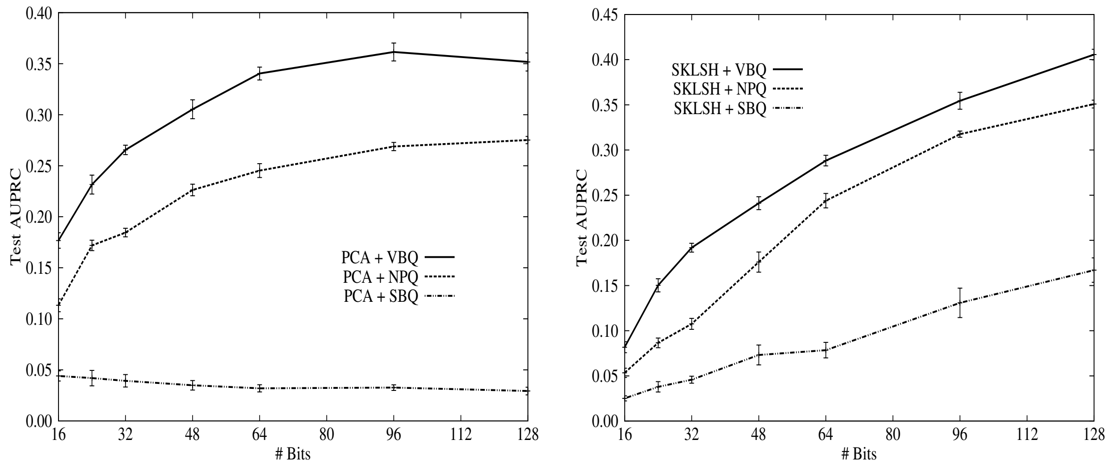
the larger NUS-WIDE dataset[4].

(a) LSH, PCA, SH, SKLSH Projections Quantised with Variable Thresholds

In this section I will examine the retrieval results obtained from quantising LSH, PCA, SH and SKLSH projections with a variable threshold allocation. The variable threshold quantisation results for ITQ projections are presented in the next section. The query-by-example image retrieval results for this section are presented in Tables 5.5-5.7 for a hashcode length of 32 bits on the CIFAR-10, LabelMe and NUS-WIDE image datasets. In addition, Figures 5.3-5.5 present the retrieval results for hashcodes of length 16-128 bits for both PCA and SKLSH projections across the three considered datasets.

The retrieval results presented in Tables 5.5-5.7 suggest that a variable allocation of thresholds (VBQ$_{bound}$, VBQ$_{greedy}$) generally outperforms a uniform allocation (NPQ, NPQ$_{opt}$) for the four considered projection functions. For example, for SKLSH projections on CIFAR-10, VBQ$_{bound}$ achieves a 71% relative increase in AUPRC compared to NPQ$_{opt}$. This increase in retrieval effectiveness is statistically significant based on a Wilcoxon signed rank test ($p < 0.01$). The results from quantising PCA projections with variable thresholds are also particularly encouraging. For example, on the CIFAR-10 image dataset VBQ$_{bound}$ obtains a statistically significant 48% relative increase at

[4]These settings of N_{trd} for CIFAR-10 and NUS-WIDE were found to be optimal in Chapter 4, Section 4.3.3.1.



(a) **PCA projections** (b) **SKLSH projections**

Figure 5.3: AUPRC versus hashcode length on CIFAR-10 for PCA (Figure (a)) and SKLSH (Figure (b)) projections. Retrieval results are shown for VBQ$_{bound}$ and NPQ$_{opt}$. Bars represent the standard error of the mean.

32 bits compared to the uniform threshold allocation of NPQ$_{opt}$ for PCA projections. The gain in retrieval effectiveness for SKLSH and PCA projections are also observed on the LabelMe (Table 5.6) and NUS-WIDE (Table 5.7) datasets. Furthermore, in Figures 5.3-5.5 it is readily apparent that this boost in retrieval effectiveness for PCA and SKLSH projections is maintained for both shorter (< 32 bits) and longer (> 32 bits) hashcodes. The same observation is made for PCA and SKLSH projections on the LabelMe (Figure 5.4) and NUS-WIDE datasets (Figure 5.5).

In contrast, the retrieval results for SH and LSH projections are less consistent across the three image datasets. Significant gains in retrieval effectiveness are found for LSH across the CIFAR-10 and LabelMe datasets, but not for the NUS-WIDE dataset. Similarity, for SH I observe a statistically significant increase in effectiveness for CIFAR-10 and NUS-WIDE, but not for the LabelMe dataset. This result suggests that a variable allocation of thresholds is not always guaranteed to give a performance boost, and in some cases appears to be projection and dataset specific. The gain in AUPRC for Spectral Hashing (SH) is perhaps slightly surprising given that this projection function itself allocates more bits to high variance directions in the feature space[5]. The fact that VBQ$_{bound}$ and VBQ$_{greedy}$ can, in some cases, extract a further gain in retrieval performance suggests that there is additional scope for improvement in retrieval effectiveness over and above the variable bit allocation mechanism of SH.

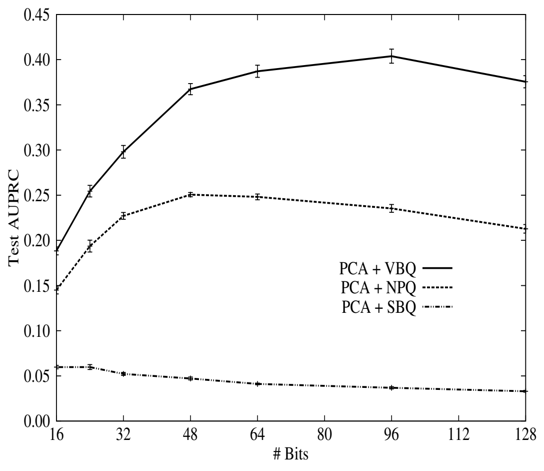
[5]The reader is referred to Chapter 2, Section 2.6.3.2 for an overview of Spectral Hashing (SH).

	Quantisation Model				
	VBQ$_{bound}$	**VBQ$_{greedy}$**	**NPQ**	**NPQ$_{opt}$**	**SBQ**
LSH	**0.2155▲**	0.2118	0.1810	0.1787	0.1574
ITQ	0.3077▼▼	0.3014	0.3658	**0.3821**	0.2822
SH	0.2500	0.2487	0.2471	**0.2581**	0.0901
PCA	0.2975	**0.3043▲▲**	0.2151	0.2306	0.0515
SKLSH	**0.2108▲▲**	0.2043	0.1311	0.1443	0.0355

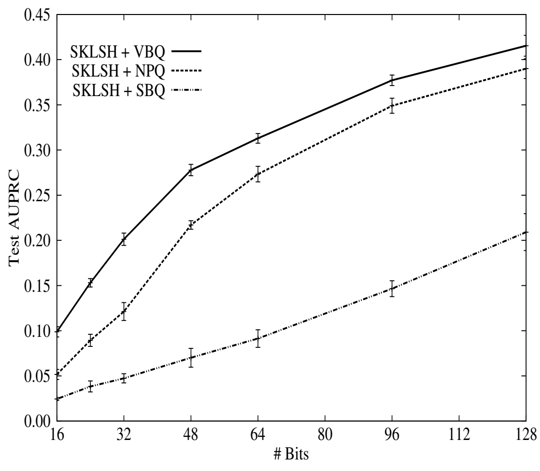
Table 5.6: AUPRC on the LabelMe dataset with a hashcode length of 32 bits. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over NPQ$_{opt}$. ▲/▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.05$) over NPQ$_{opt}$

Lastly, comparing the optimised variant of my NPQ model (NPQ$_{opt}$) to the un-optimised variant (NPQ) in Tables 5.5-5.7 I note that there is a small boost in retrieval effectiveness across the three considered datasets. This result suggests that $\beta = 1$ is generally a suitable setting for NPQ, and varying β is only necessary for VBQ when using the F_β-measure as the basis for computing an optimal threshold allocation across projected dimensions. Furthermore, the results in Table 5.5 indicate that there is no significant difference between the AUPRC scores resulting from the literature standard and improved splitting strategies outlined in Chapter 3, Section 3.5. This latter observation agrees with the wealth of similar findings presented in Chapter 4, Section 4.3.3.5.

In conclusion, based on these results, I can confirm my primary hypothesis (H_1) that a variable allocation of thresholds can indeed be beneficial to the effectiveness of nearest neighbour search using many different projection functions, both of the data-dependent and independent variety. The results in this section strongly suggest that my supervised scoring metric (the F_β-measure) is effective at downweighting the contribution of ineffective hyperplanes to the computation of the hashcode ranking metric (Manhattan distance). The downweighting is achieved by allocating a low number of thresholds (or none) to these more ineffective projected dimensions, which in turn will reduce the amount of bits contributed from these dimensions to the hashcode for a data-point.



(a) **PCA projections**



(b) **SKLSH projections**

Figure 5.4: AUPRC versus hashcode length on LabelMe for PCA (Figure (a)) and SKLSH (Figure (b)) projections. Retrieval results are shown for VBQ$_{bound}$ and NPQ$_{opt}$. Bars represent the standard error of the mean.

(b) ITQ Projections Quantised with Variable Thresholds

ITQ is the only projection function on which a variable allocation of thresholds fails to achieve an increase in retrieval effectiveness over a uniform allocation with my multi-threshold quantisation model (NPQ) presented in Chapter 4. In fact, when compared to NPQ, both VBQ$_{bound}$ and VBQ$_{greedy}$ appear to hurt retrieval effectiveness when quantising ITQ projections (18% decrease in AUPRC versus NPQ$_{opt}$). Recall from my review in Chapter 2, Section 2.6.3.3 that ITQ rotates the input feature space to balance the variance captured by the K hyperplanes. The assumption made by ITQ is that hyperplanes capturing a low amount of the variance in the input feature space generate poor quality bits as they generally fail to clump related data-points close together along a projected dimension. By rotating the input feature space the variance captured becomes more balanced across the K hyperplanes and each corresponding bit therefore generally has an equal quality. I conjecture that the rotation performed by ITQ is removing part of the signal relied upon by my variable threshold allocation algorithms, namely a high degree of difference between the locality preserving qualities of the K hyperplanes. Recall that my variable threshold allocation algorithms specifically seek out informative hyperplanes in order to allocate those hyperplanes proportionally more thresholds than the more uninformative hyperplanes. With this signal evidently all but removed the variable threshold allocation struggles to improve beyond a uniform multi-threshold allocation (NPQ). The fact that VBQ$_{bound}$ and VBQ$_{greedy}$ exhibit

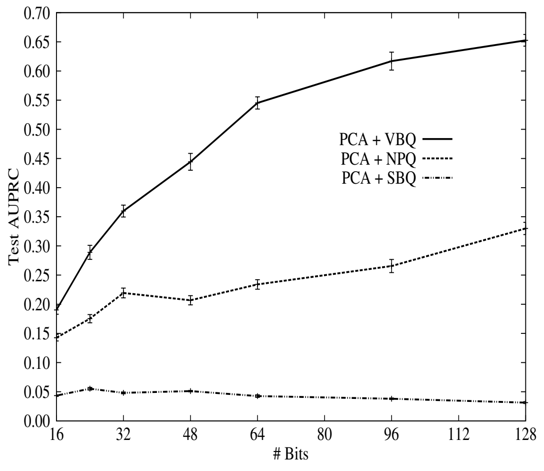
	Quantisation Model				
	VBQ$_{bound}$	**VBQ$_{greedy}$**	**NPQ**	**NPQ$_{opt}$**	**SBQ**
LSH	**0.5119**	0.4970	0.4238	0.4507	0.2961
ITQ	0.4438▼▼	0.4438	0.5130	**0.5226**	0.4842
SH	**0.3323▲▲**	0.3251	0.1965	0.1970	0.0232
PCA	**0.3741▲▲**	0.3621	0.2178	0.2340	0.0516
SKLSH	0.4505	**0.4675▲▲**	0.2650	0.2798	0.0310

Table 5.7: AUPRC on the NUS-WIDE dataset with a hashcode length of 32 bits. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over NPQ$_{opt}$.

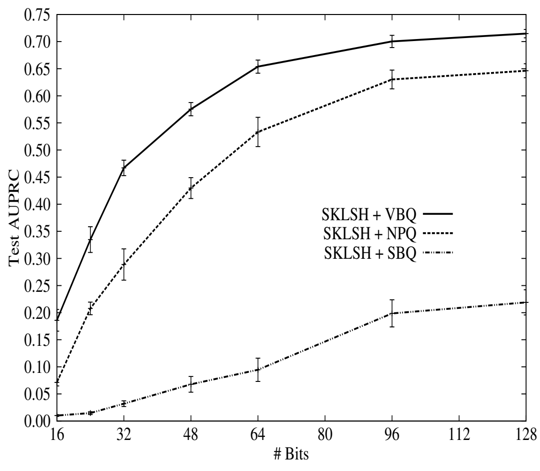
a lower AUPRC than NPQ can be explained by the different allocations of thresholds made by the algorithms. In the case of NPQ I manually specified an allocation of three thresholds (2 bits) per projected dimension and therefore $K/2$ of the available hyperplanes were discarded. On the other hand, VBQ$_{bound}$ and VBQ$_{greedy}$ chose to allocate only one threshold (1 bit) per projected dimension and therefore retained all K of the available hyperplanes, which is clearly not optimal in this case. Allocation of thresholds based on F_β-measure maximisation is therefore not perfectly correlated with maximisation of AUPRC.

(c) Examining the Bit Allocations of VBQ$_{bound}$

Finally, it is interesting to examine the specific allocation of bits (thresholds) made by VBQ$_{bound}$. I show in Figure 5.6 the bit allocation assigned by VBQ$_{bound}$ to each of the 32 hyperplanes generated by the PCA and SKLSH projection functions. In Figure 5.6a I rank the PCA projected dimensions in descending order of variance, so that the first projected dimension has the highest overall variance, the second projected dimension exhibits the second highest variance and so forth. It is clear that the higher variance PCA hyperplanes (1-10) garner the vast proportion of the available bit budget (equivalently thresholds). This result suggests that those hyperplanes that capture the highest variance in the input feature space are also effective at preserving the pairwise relationships between data-points encoded in the adjacency matrix **S**. Nevertheless, I also note that lower variance hyperplanes also attract a proportion of the available bits (12, 14, 19, 22), which suggests that variance is not perfectly correlated with neighbourhood preservation. This latter fact is not surprising given that variance is a quantity computed in an unsupervised manner, independent of any knowledge on



(a) **PCA projections**



(b) **SKLSH projections**

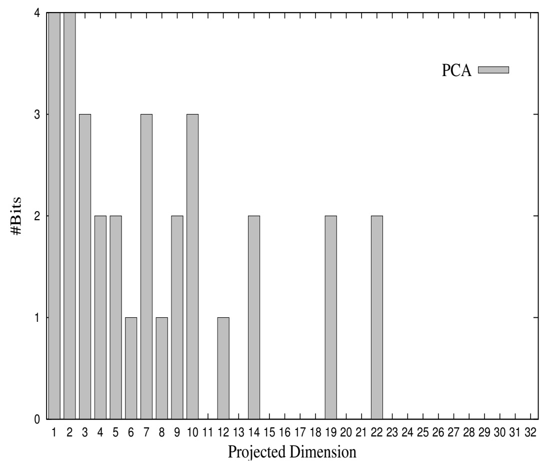
Figure 5.5: AUPRC versus hashcode length on NUSWIDE for PCA (Figure (a)) and SKLSH (Figure (b)) projections. Retrieval results are shown for VBQ$_{bound}$ and NPQ$_{opt}$. Bars represent the standard error of the mean.

the true data-point relationships, whereas my F_β-measure explicitly seeks out the most locality preserving hyperplanes using supervision from the adjacency matrix **S**. In Figure 5.6b I show the allocation for the SKLSH projected dimensions. In contrast to PCA, the bit budget is more evenly allocated across the 32 projected dimensions with a large proportion of the dimensions attracting one bit each with a smaller number of hyperplanes attracting two bits compared to PCA. This result is quite intuitive if we consider that SKLSH draws the hyperplanes randomly within the input feature space.

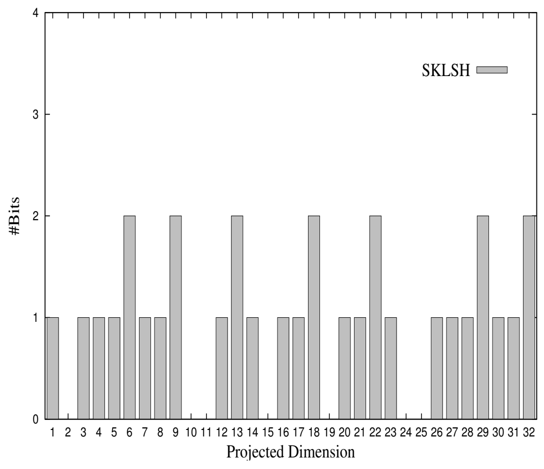
An attractive property of my variable threshold algorithms is the pruning that is performed on hyperplanes that exhibit a low locality preservation. Notice in Figure 5.6a that 18 out of 32 (56%) of the available PCA hyperplanes have been discarded (assigned 0 bits). This sparse solution is a form of dimensionality reduction that may be particularly advantageous to end-applications where there is limited memory to store potentially high dimensional and dense hyperplane normal vectors (for example, in embedded systems).

5.3.3.3 Experiment III: Branch-and-Bound versus Greedy Allocation

In this section I examine the second and final hypothesis of this chapter, namely that finding a threshold allocation using a branch-and-bound search (VBQ$_{bound}$) leads to a higher retrieval effectiveness than the greedy threshold redistribution algorithm VBQ$_{greedy}$. The retrieval results on CIFAR, LabelMe and NUS-WIDE are shown in



(a) **PCA projections**



(b) **SKLSH projections**

Figure 5.6: The bit allocation assigned by VBQ$_{bound}$ for PCA (Figure (a)) and SKLSH projections (Figure (b)).

Tables 5.5-5.7. I conduct a Wilcoxon signed rank test to determine whether the difference in the AUPRC values for VBQ$_{bound}$ and VBQ$_{greedy}$ obtained over ten random dataset splits is significant. On the CIFAR-10 dataset for all considered projection functions (ITQ, PCA, LSH, SH, SKLSH) I find that the difference between the AUPRC achieved by both algorithms is *not* significant with $p < 0.01$. I find a similar result on the LabelMe and NUS-WIDE datasets for all considered projection functions. Taken together these results suggest that I *cannot* refute the null hypothesis and therefore I am not able to confirm hypothesis H_2 based on the experiments conducted in this section. From an efficiency standpoint this is an encouraging finding because it means we can benefit from the substantially faster training and allocation time of the VBQ$_{greedy}$ model while attaining a retrieval effectiveness that is statistically indistinguishable to the more computationally expensive VBQ$_{bound}$.

5.3.3.4 Experiment IV: Evaluation of Training Time

In this last experiment I will examine the training time of my variable threshold allocation models and compare the computational cost to the multi-threshold quantisation algorithm introduced in Chapter 4. The timing results in seconds are given in Table 5.8. I break the computation time into the time taken to *learn the thresholds* and the time taken to compute the *threshold allocation*. The multi-threshold quantisation algorithm (NPQ) introduced in Chapter 4 attains the lowest training time, which is not unexpected given it only learns thresholds for one particular threshold quantity (T) and

	Threshold Learning	**Time (s)**	**Threshold Allocation**	**Time (s)**
VBQ$_{bound}$	$O(T_{max}N_{trd}^2F + KN_{trd}^2)$	42.6	$O(2^{(B_{max}+1)K})$	0.134
VBQ$_{greedy}$	$O(T_{gdy}N_{trd}^2F + KN_{trd}^2)$	30.0	$O(1)$	0.001
NPQ	$O(K^{'}TN_{trd}^2F)$	3.60	–	–

Table 5.8: Threshold learning and allocation timings for the proposed models against NPQ. Time (seconds) taken at 32 bits (CIFAR-10, LSH) to grade the hyperplanes (*learning thresholds*) and the time then taken to solve the threshold allocation problem (*threshold allocation*). $T_{max} = K\sum_{b=1}^{B_{max}} 2^b - 1$ is an expression for the total number of thresholds, with $K^{'}T \ll T_{gdy} \ll T_{max}$, where the factor T_{gdy} depends on the specific choices made by the greedy algorithm during a particular run. $K^{'} = \lfloor K/B \rfloor$ denotes the number of hyperplanes used to generate K-bits for NPQ, with a constant allocation of B bits per projected dimension. N$_{trd}$ denotes the number of training data-points, F the number of objective function evaluations (Equation 5.6). The timing results were recorded on an otherwise idle Intel 2.7GHz, 16Gb RAM machine and averaged over 10 random dataset partitions. All models are implemented in the same software stack (Matlab).

has no need to learn the allocation. In terms of threshold allocation time the VBQ$_{greedy}$ threshold allocation algorithm (Section 5.2.2.4) is *two orders* of magnitude faster than the binary integer linear program (VBQ$_{bound}$). Threshold learning is 30% faster for VBQ$_{greedy}$ compared to VBQ$_{bound}$ due to the greedy on-demand computation of the F_β-measure scores which ensures that there is no wasted computation.

5.4 Conclusions

In this chapter I have proposed two new algorithms for learning a *variable allocation* of quantisation thresholds per projected dimension. The intuition behind these quantisation algorithms was that the locality preserving quality of the hashing hyperplanes tends to vary widely within the input feature space. My central argument was that those hyperplanes that are better at maintaining the neighbourhood structure between the data-points should attract a higher quantity of the available thresholds, and vice-versa for lower quality hyperplanes. The quantity of thresholds assigned to a given projected dimension dictates how finely that dimension is quantised, with a greater allocation of thresholds contributing to a much less granular partitioning of the projected dimension. I argued in this chapter that the level of granularity, or equivalently the number of thresholds assigned per projected dimension, was an important factor in the ultimate retrieval effectiveness of the generated hashcodes. Quantising a projected dimension with a high locality preserving power with too few thresholds will run the risk of grouping together many unrelated data-points in the same quantised regions, therefore leading to a high number of false positives. To capitalise on the additional structure available in the higher quality projected dimensions, a higher number of thresholds should ideally be assigned to that dimension so that as many related data-points fall within the same quantised regions while minimising the number of unrelated data-points in the same regions. However, a quantisation of too fine a granularity may also result in a high degree of false negatives, that is many true nearest neighbours falling within different quantised regions. My contention was that there is a *sweet spot* for the number of thresholds that minimises the number of false positives and false negatives for a given projected dimension.

To learn the optimal number of thresholds I made two key contributions in this chapter: firstly in Section 5.2.2.1, I proposed the F_β-measure as a metric for grading the neighbourhood preserving quality of the projected dimensions and, secondly, in Sections 5.2.2.3-5.2.2.4 I introduced two new algorithms that used this scoring function to compute an allocation of thresholds subject to a total threshold budget. The variable threshold allocation algorithms tackled this NP-hard optimisation problem using two different strategies. My first proposed algorithm formulated the allocation problem as a binary integer linear programme (BILP) which was solved using branch and bound (Section 5.2.2.3). My alternative allocation algorithm greedily redistributed thresholds from the lowest quality hyperplanes to the highest quality hyperplanes (Section 5.2.2.4). To the best of my knowledge the research in this chapter is the first to introduce and solve the variable threshold allocation problem in the context of hashing-based ANN search.

I evaluated the proposed variable threshold quantisation algorithms in an extensive set of image retrieval experiments against the state-of-the-art baseline model introduced in Chapter 4. The key findings from the experimental evaluation were three-fold:

- Allocating a variable number of quantisation thresholds per projected dimension using the F_β-measure as an indicator of hyperplane quality was shown, for certain projection functions, to lead to higher quality hashcodes and a significantly higher retrieval effectiveness compared to a uniform allocation of thresholds.

Section 5.3.3.2 and Tables 5.5-5.7 present the experimental results that support this claim.

- There is no significant difference (Wilcoxon signed rank test, $p < 0.01$) in retrieval effectiveness between a branch-and-bound based algorithm for solving the threshold allocation and a greedy threshold redistribution algorithm. The latter allocation algorithm is however *two orders* of magnitude faster at threshold allocation time and 28% *faster* at threshold learning. The quantitative results supporting this claim are presented in Section 5.3.3.4 and Table 5.8.

- There is no significant difference (Wilcoxon signed rank test, $p < 0.01$) between the retrieval results originating from the literature standard and improved dataset splitting strategies first outlined in Chapter 3, Section 3.5. This result accords with similar findings from my experiments in Chapter 4. Section 5.3.3.2 and Table 5.5 presents the results that validate this claim.

To the Computer Vision practitioner, the findings of this chapter hint at the following recommendations, dependent on whether an eigendecomposition is possible on the dataset of interest:

- If it is computationally tractable to compute a PCA projection on the dataset then a rotation of the feature space (ITQ) followed by the uniform multiple threshold quantisation algorithm of Chapter 4 leads to the overall highest retrieval effectiveness.

- If computationally constrained to other projection functions, such as LSH/SKLSH, then a variable threshold quantisation is highly advantageous to retrieval effectiveness compared to a uniform allocation. It is most likely intractable to compute PCA on the large high-dimensional datasets prevalent in real-world nearest neighbour search scenarios, and so this use-case of being constrained to random projections is perhaps the most common. To learn the thresholds and allocation the greedy approach (VBQ$_{greedy}$) is the most efficient while still maintaining attractive accuracy.

In this chapter and the previous chapter, I have studied the process of scalar quantisation for hashing-based ANN search in considerable detail, proposing a set of new quantisation algorithms that were found to significantly improve hashcode quality and the resulting effectiveness of query-by-example image retrieval. In Chapter 6 I will now turn my attention to the related problem of *projection function learning* which fractures the input feature space with a set of K hyperplanes in a way that attempts to maximise the number of true nearest neighbours that fall within the same polytope-shaped regions of the space.

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.

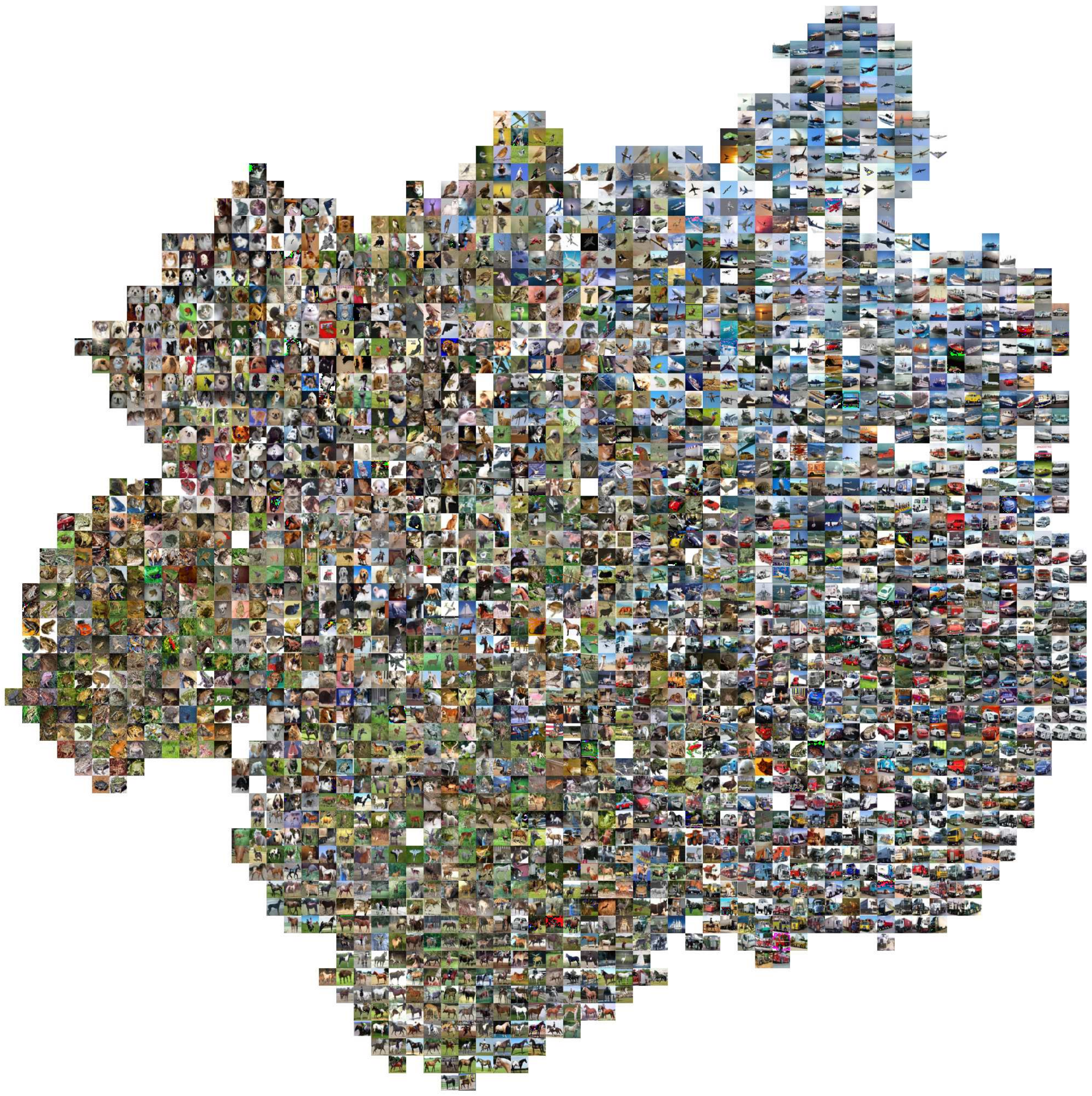


Figure 6.1: t-SNE visualisation of a subset of the CIFAR-10 image dataset (20,000 images). Image positions are computed using a 2-dimensional t-SNE projection (van der Maaten and Hinton (2008)) of a 32-dimensional embedding produced by the non-linear variant of my graph regularised projection function. 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* $\left\{\mathbf{h}_k \in \mathbb{R}^D\right\}_{k=1}^K$ with normal vectors $\left\{\mathbf{w}_k \in \mathbb{R}^D\right\}_{k=1}^K$ 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 $\mathbf{S}$.

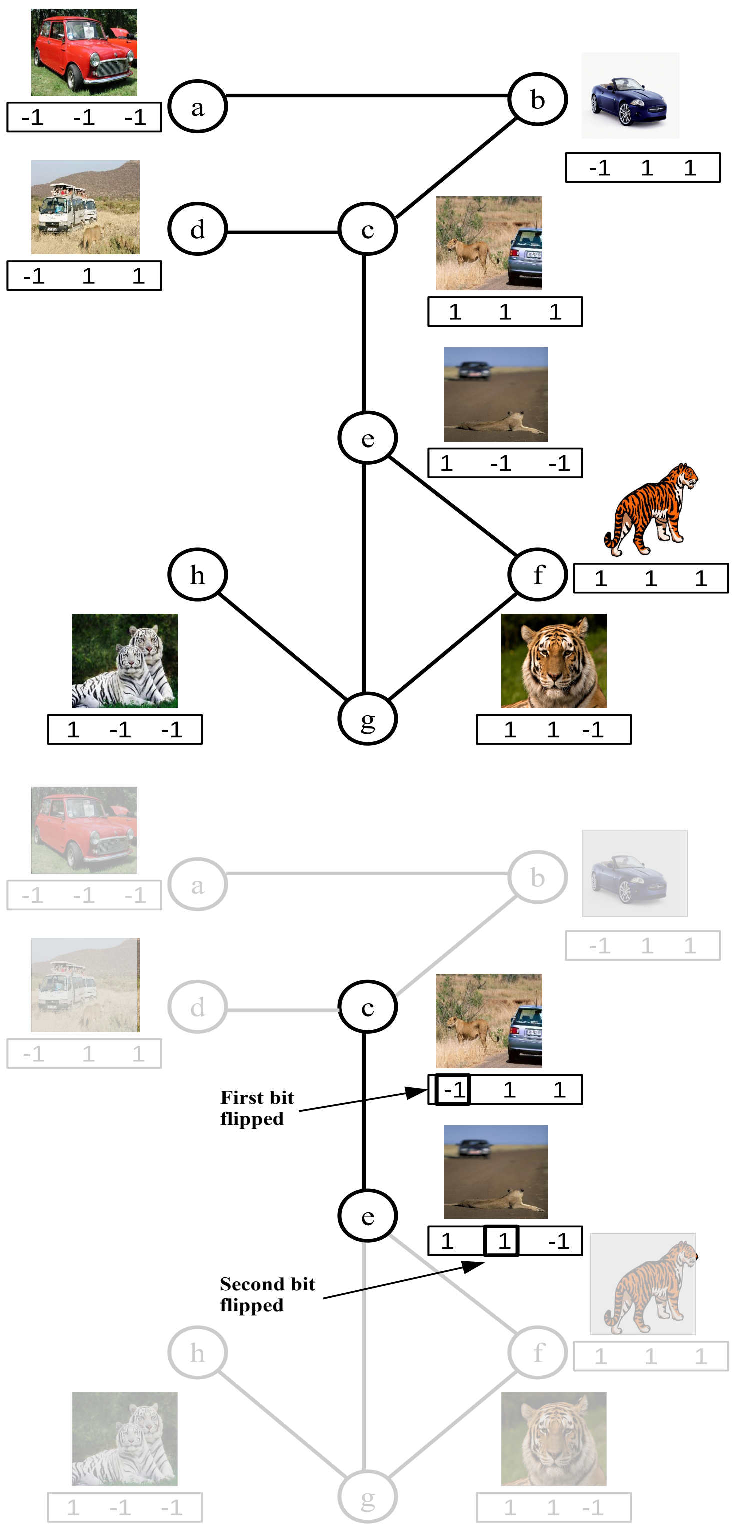


Figure 6.2: The regularisation step. Nodes represent data-points and arcs represent neighbour relationships. The 3-bit hashcode assigned to a given node is shown in the boxes. Top: The original hashcode assignment at initialisation. Bottom: The hashcode update for nodes c and e.

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:

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

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 $\mathbf{h}_2$).

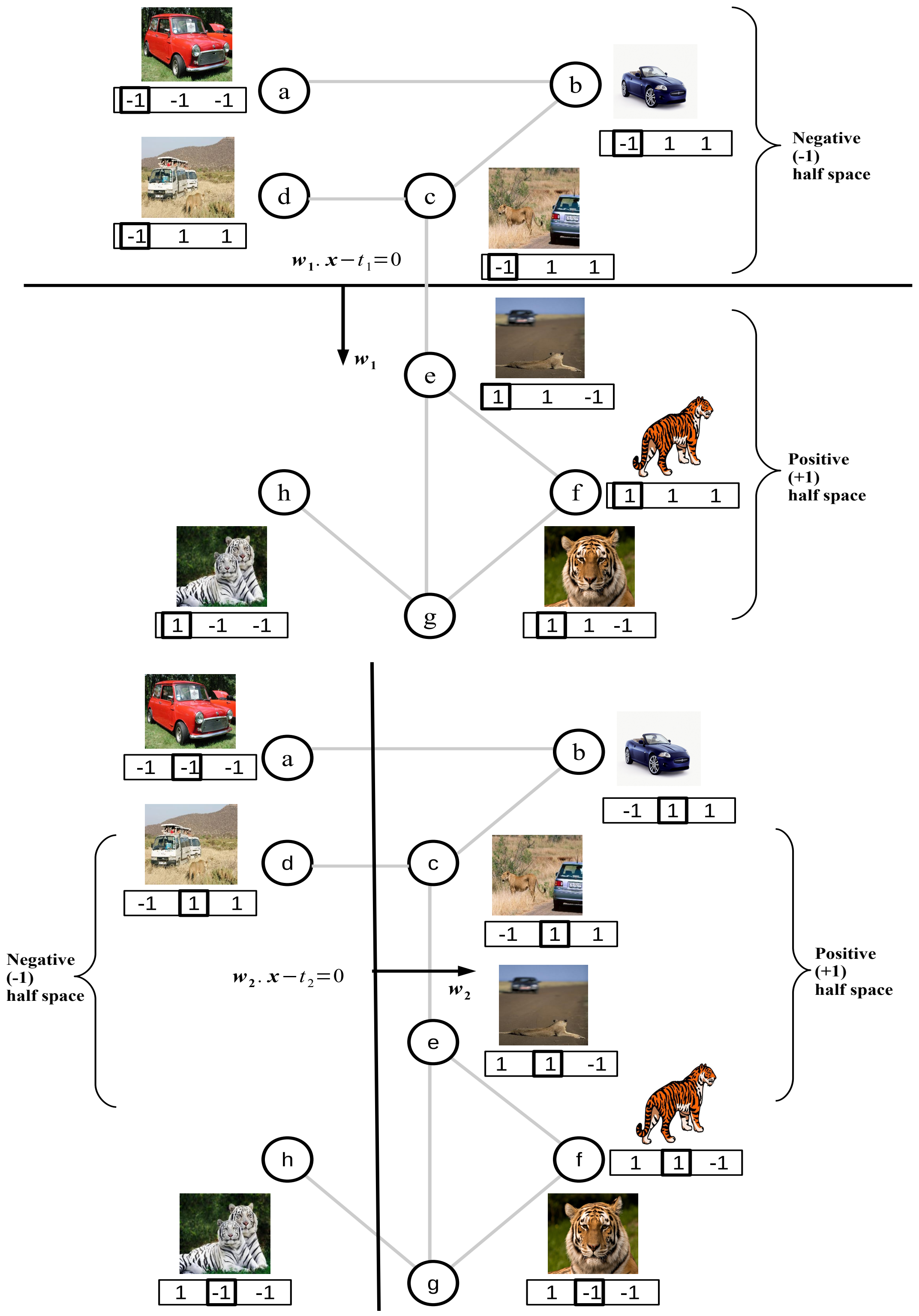


Figure 6.3: The partitioning step. In this stage, the regularised hashcodes are used to re-position the hashing hyperplanes. Top: First bit of hashcode. Bottom: Second bit.

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:

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

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 $\{t_1 \ldots t_K\}$
1  Initialise $\mathbf{B}_0 \in \{0,1\}^{N_{trd} \times K}$ via LSH or ITQ+CCA from $\mathbf{X}$
2  $\mathbf{B}_0 = \text{sgn}(\mathbf{B}_0 - \frac{1}{2})$
3  $\mathbf{B} = \mathbf{B}_0$
4  for $m \leftarrow 1$ to $M$ do
5      $\mathbf{B} = \text{sgn}\left(\alpha \mathbf{S}\mathbf{D}^{-1}\mathbf{B} + (1-\alpha)\mathbf{B}_0\right)$
6      for $k \leftarrow 1$ to $K$ do
7          $\mathbf{b}_k = \mathbf{B}(:,k)$
8          Train SVM$_k$ with $\mathbf{b}_k$ as labels, training dataset $\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D}$
9          Obtain hyperplane $\mathbf{w}_k$ and bias $t_k$
10     end
11     $B_{ik} = \text{sgn}(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k)$ for $i = \{1 \ldots N_{trd}\}$ and $k = \{1 \ldots K\}$
12 end
13 return $\left\{\mathbf{w}_k \in \mathbb{R}^D\right\}_{k=1}^K$, $\{t_k \in \mathbb{R}\}_{k=1}^K$
```

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.

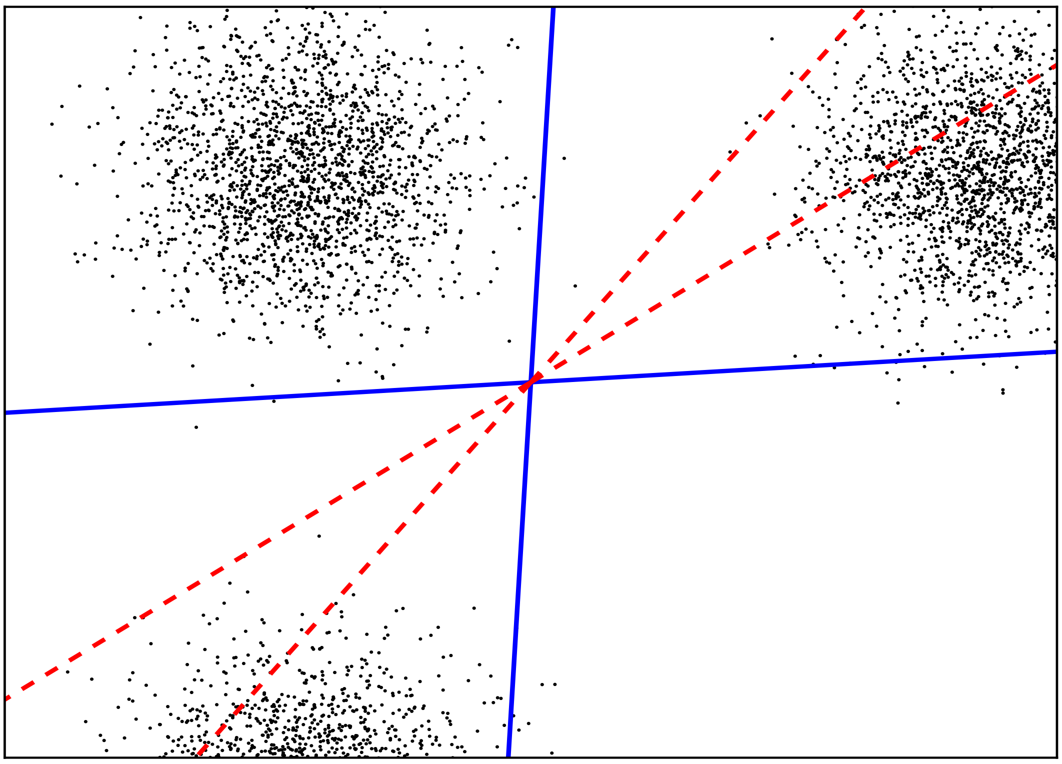


Figure 6.4: Synthetic toy example illustrating the operation of my graph regularised projection function. The toy dataset consists of 6,000 data-points clustered into three distinct clusters. The data-points in each cluster are of the same class as each other (out of three possible classes), and therefore 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 $M \in \{1, 2 \dots 19, 20\}$ and $\alpha \in \{0.1, 0.2, \dots, 0.9, 1.0\}$, selecting the overall configuration that leads to the highest *validation* dataset mAP. For a given α, to determine a suitable value for M, I

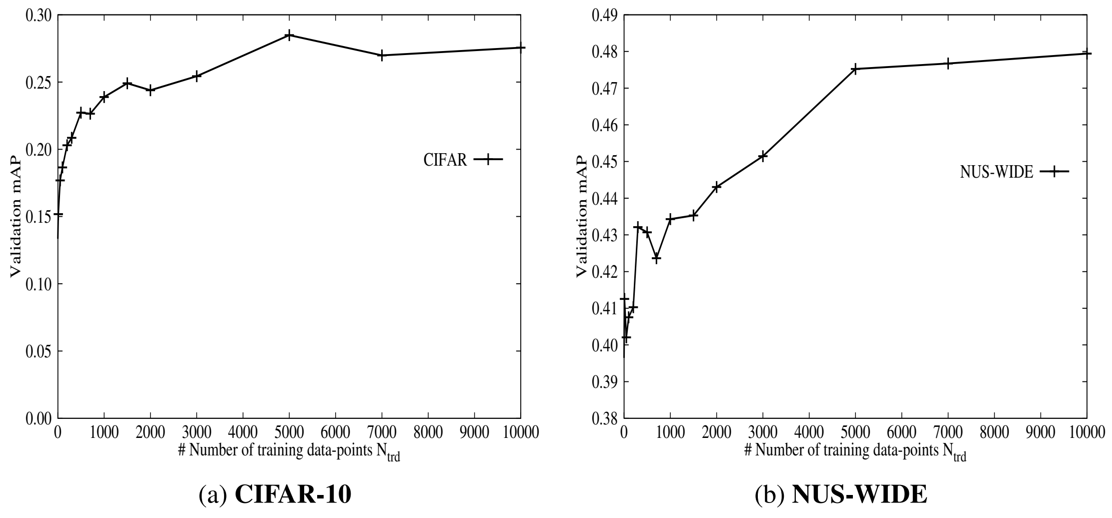


Figure 6.5: Learning curves for CIFAR-10 (Figure (a)) and NUS-WIDE (Figure (b)) as the number of training data-points N_{trd} is increased.

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 **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 $\{\mathbf{w}_k \in \mathbb{R}^D\}_{k=1}^{K}$. 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 (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

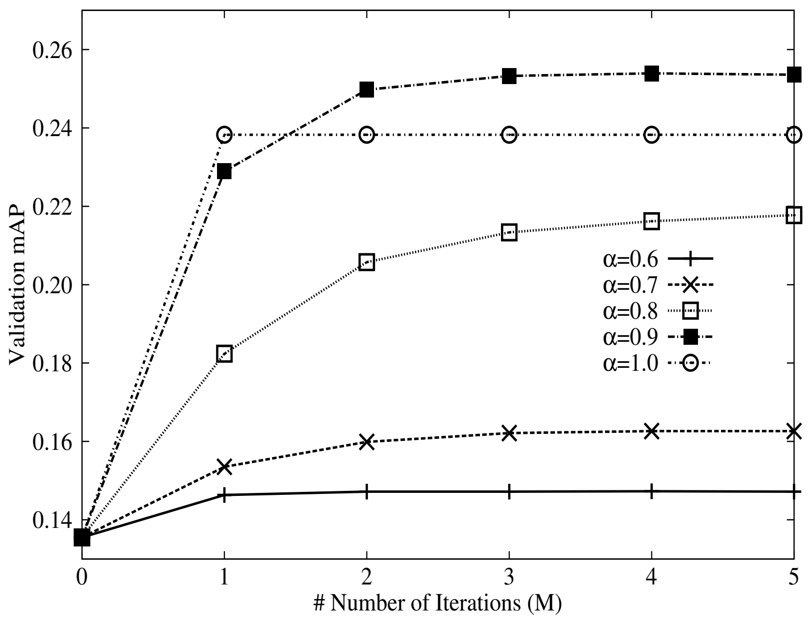
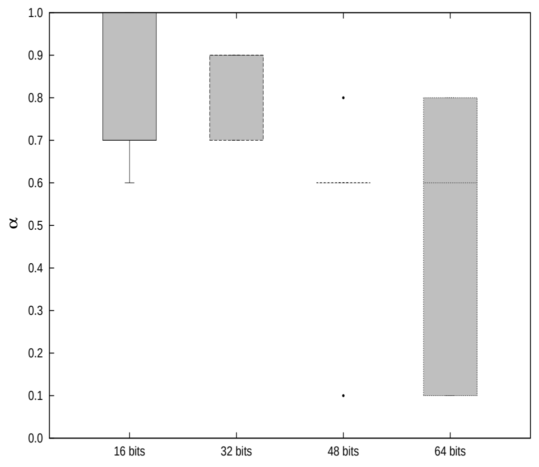


Figure 6.6: CIFAR-10 validation mAP at 32 bits versus iterations (M) and the setting of the interpolation parameter α. GRH is parameterised with a linear kernel and an LSH initialisation. The setting of α is more important to final retrieval effectiveness than the value of M.

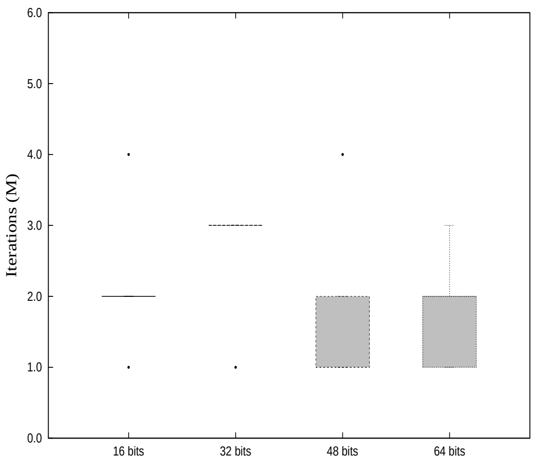
the optimal setting of both parameters is likely to be tied given their existence within the central optimisation loop of my algorithm. To isolate the effect of α and M only, I keep the linear SVM flexibility of margin set to $C = 1.0$ throughout this experiment.

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 procedure 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.

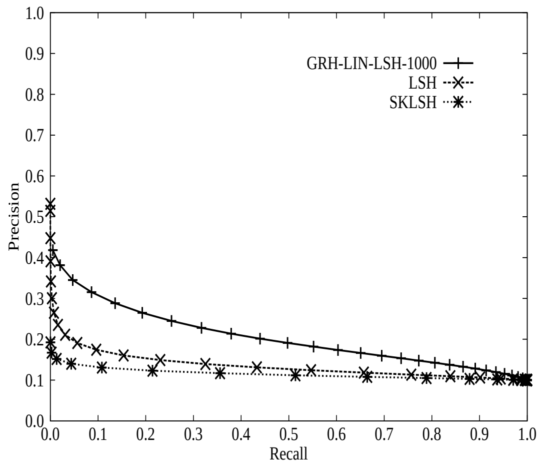
NUS-WIDE (Experiment I)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
LSH	0.3837	0.3883	0.3863	0.3879
SKLSH	0.3724	0.3755	0.3734	0.3827
GRH$_{lin,lsh}$	**0.4928▲▲**	**0.4971▲▲**	**0.5023▲▲**	**0.5065▲▲**

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.

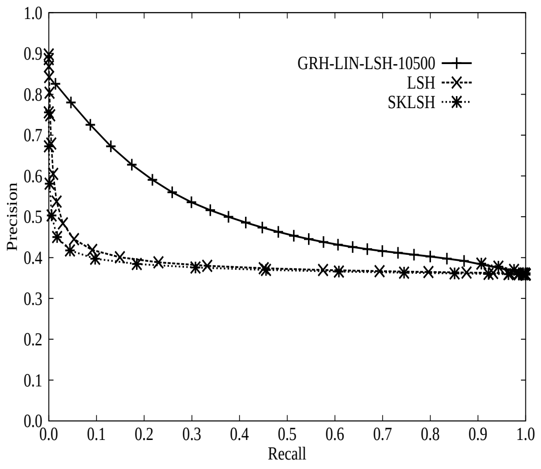
SIFT1M (Experiment I)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
LSH	0.0273	0.1051	0.2023	0.2524
GRH$_{lin,lsh}$	**0.0388▲**	**0.1569▲▲**	**0.2980▲▲**	**0.3860▲▲**

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 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,

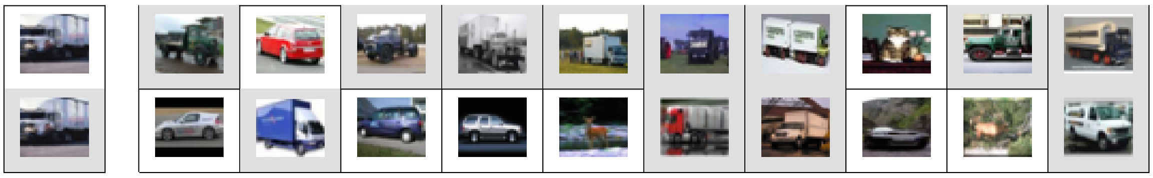


Table 6.9: **Left-most column:** *Truck* query image. **Top row:** GRH top 10 retrieved images, precision: 0.8. **Bottom row:** LSH top 10 retrieved images, precision: 0.4. Shaded cells indicated true positives.

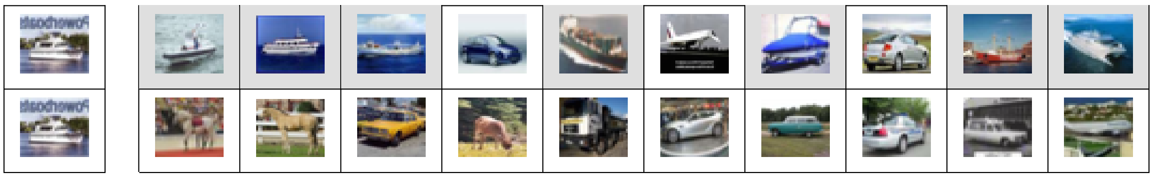


Table 6.10: **Left-most column:** *Boat* query image. **Top row:** GRH top 10 retrieved images, precision: 0.5. **Bottom row:** LSH top 10 retrieved images, precision: 0.0.

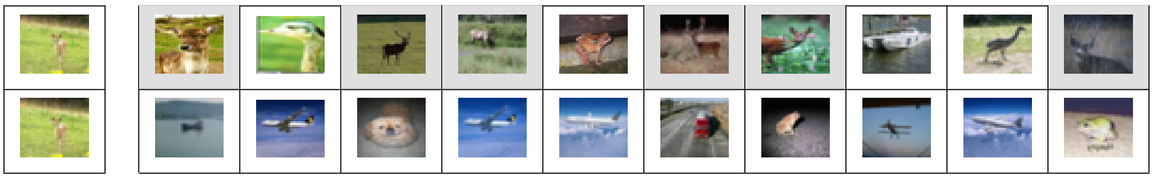


Table 6.11: **Left-most column:** *Deer* query image. **Top row:** GRH top 10 retrieved images, precision: 0.5. **Bottom row:** LSH top 10 retrieved images, precision: 0.0.

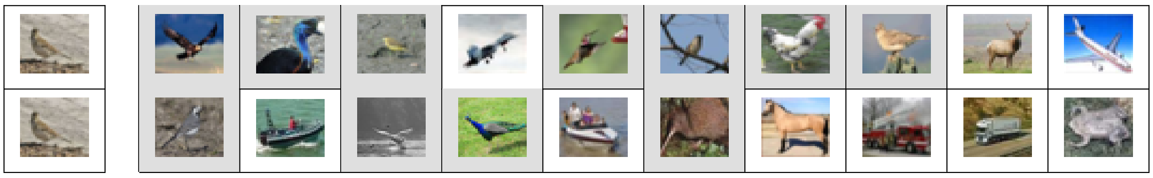


Table 6.12: **Left-most column:** *Bird* query image. **Top row:** GRH top 10 retrieved images, precision: 0.7. **Bottom row:** LSH top 10 retrieved images, precision: 0.4.

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▲▲**

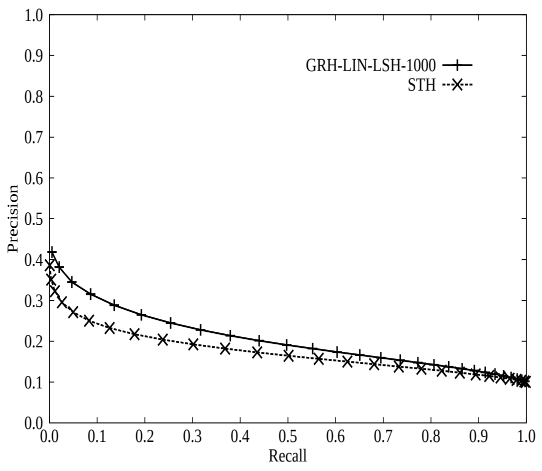
Table 6.15: mAP on the CIFAR-10 image dataset. lin: linear kernel, lsh: LSH initialisation. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over STH.

or LapEig.

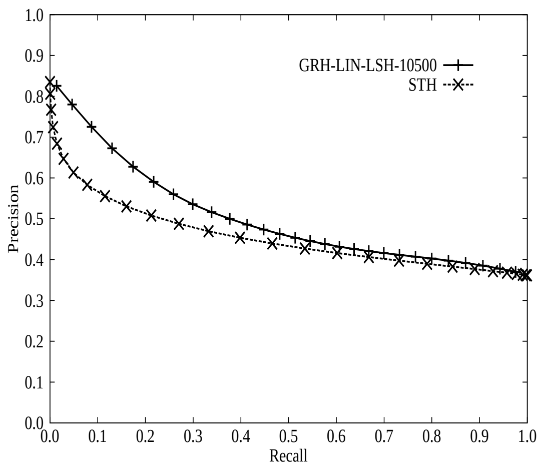
To ensure a fair comparison between both models I use the same data-point adjacency graph $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}, N_{trd} = 1{,}000$ for both STH and my own model. I also tune the linear SVM parameters of STH in the same way I tune the linear SVM parameters of my own model (Section 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 (GRH$_{lin,cca}$) yields a significantly higher retrieval effectiveness (Wilcoxon signed rank test, $p < 0.01$) than the unsupervised initialisation (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**
GRH$_{lin,lsh}$	0.2149	0.2443	0.2433	0.2460
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 GRH$_{lin,lsh}$.

NUS-WIDE (Experiment IV)				
	16 bits	**32 bits**	**48 bits**	**64 bits**
GRH$_{lin,lsh}$	0.4928	0.4971	0.5023	**0.5065**
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.

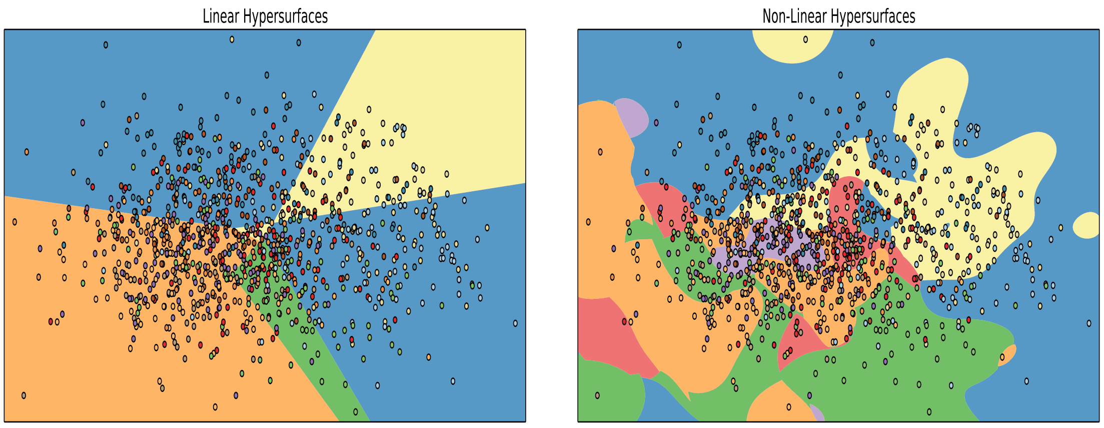


Figure 6.10: The differences between the decision boundaries induced by linear hypersurface (left) and non-linear hypersurfaces (right) on a 2D PCA projection of the CIFAR-10 dataset with 10 classes. Data-points and their classes are indicated by the coloured dots. Regions are coloured according to the 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**
GRH$_{lin,cca}$	0.4969	0.4985	0.5027	0.5000
GRH$_{rbf,full,cca}$	**0.4996**	**0.5144**	**0.5217**	**0.5269**

Table 6.20: 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 $\{t_1 \ldots t_K\}$, $\{o_1 \ldots o_K\}$
 1 Initialise $\mathbf{B}_0 \in \{0,1\}^{N_{trd} \times K}$ via LSH or ITQ+CCA from $\mathbf{X}$
 2 $\mathbf{B}_0 = \mathrm{sgn}(\mathbf{B}_0 - \frac{1}{2})$
 3 $\mathbf{B} = \mathbf{B}_0$
 4 for $m \leftarrow 1$ to $M$ do
 5     $\mathbf{B} = \mathrm{sgn}\left(\alpha \mathbf{S}\mathbf{D}^{-1}\mathbf{B} + (1-\alpha)\mathbf{B}_0\right)$
 6     for $k \leftarrow 1$ to $K$ do
 7         $\mathbf{b}_k = \mathbf{B}(:,k)$
 8         Train SVM$_k^x$ with $\mathbf{b}_k$ as labels, training dataset $\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D_x}$
 9         Train SVM$_k^z$ with $\mathbf{b}_k$ as labels, training dataset $\mathbf{Z}_{trd} \in \mathbb{R}^{N_{trd} \times D_z}$
10         Obtain hyperplanes $\mathbf{w}_k, \mathbf{u}_k$ and biases $t_k, o_k$
11     end
12     $B_{ik} = \mathrm{sgn}(\mathbf{w}_k^\intercal \mathbf{x}_i + t_k)$ for $i = \{1 \ldots N_{trd}\}$ and $k = \{1 \ldots K\}$
13 end
14 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).

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

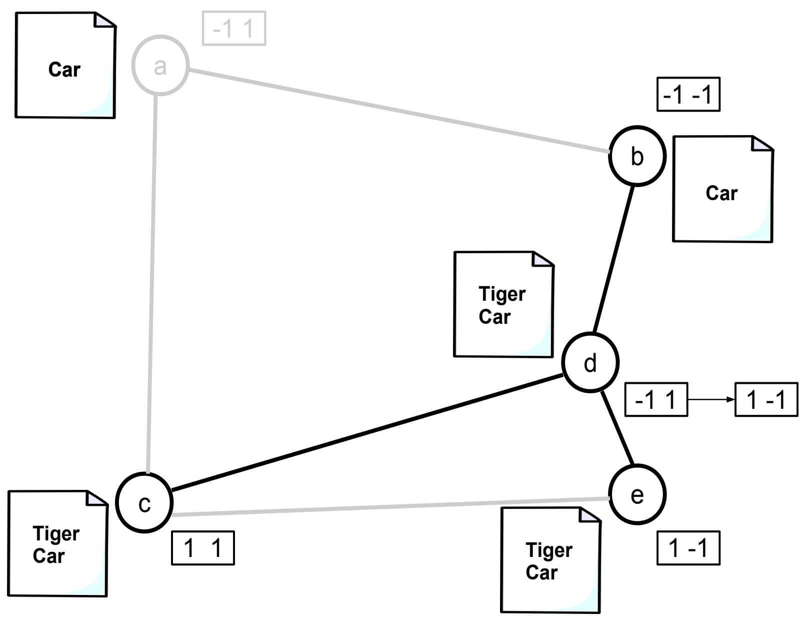
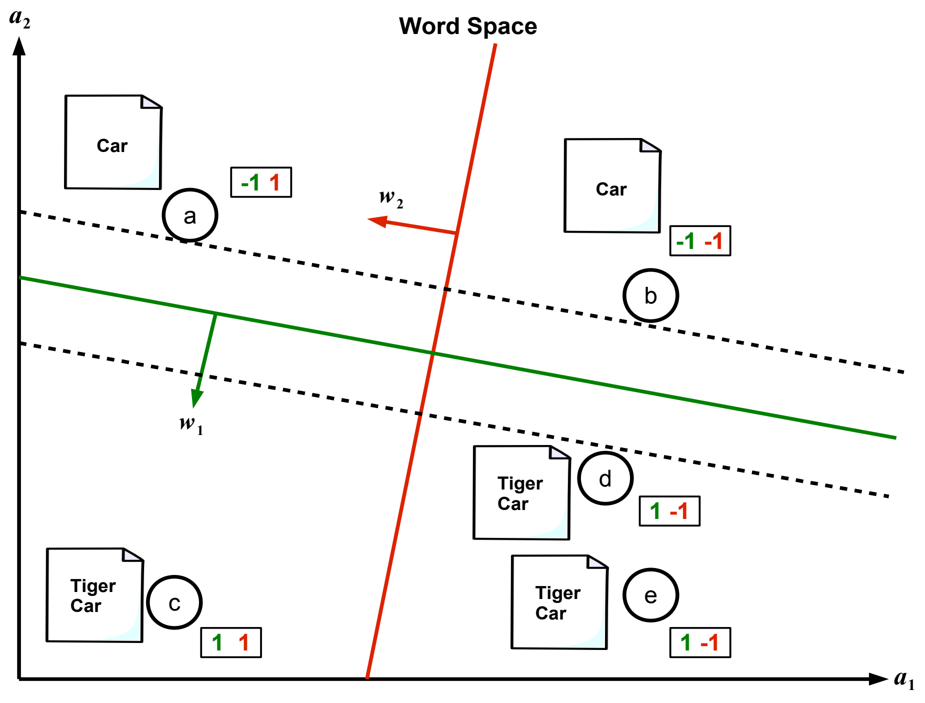


Figure 6.11: Regularisation step: the hashcode for annotation node d is updated to be more similar with its neighbours (c,b,e).

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 $\mathbf{u}_2$.



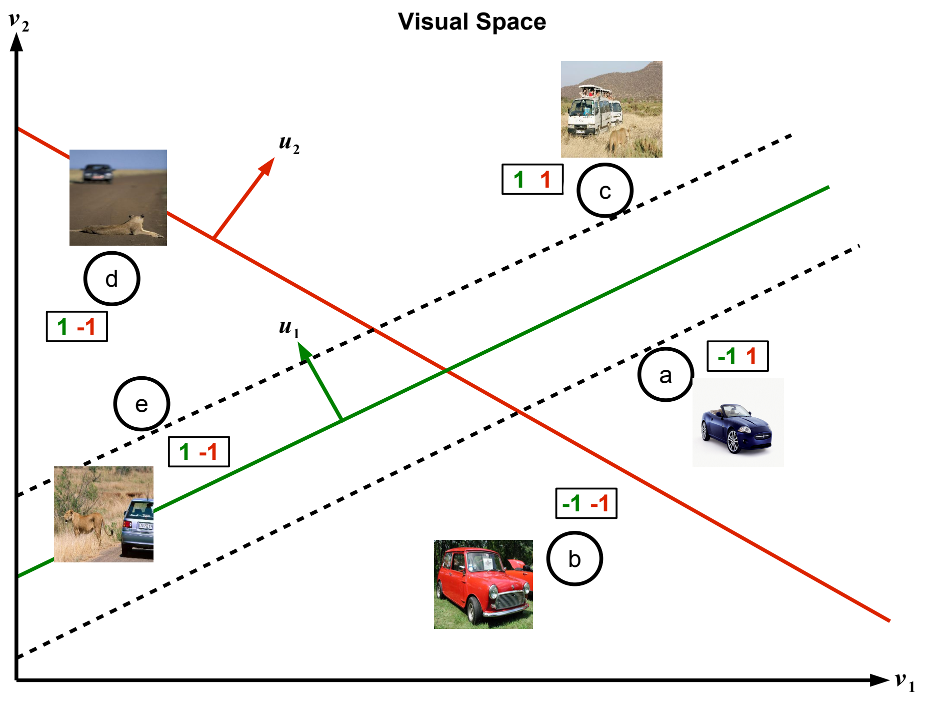


Figure 6.12: Partitioning step: hyperplanes are learnt in the annotation (top) and visual (bottom) space using annotation bits as labels. Hyperplanes in both feature spaces are positioned in such a way that nodes with the same letter are assigned the same bits in both feature spaces.

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

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

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

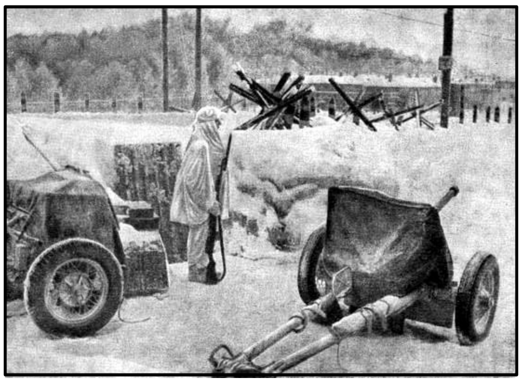
Table 6.27: Training and testing time (seconds) on the Wiki dataset at 24 bits. RCMH is parameterised with $M = 1, M^{'} = 5$ and $N_{trd} = 574, N_{teq} = 574, D_x = 128, D_z = 10$. The timing results were recorded on an otherwise idle Intel 2.7GHz, 16Gb RAM machine and averaged over ten runs. All models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

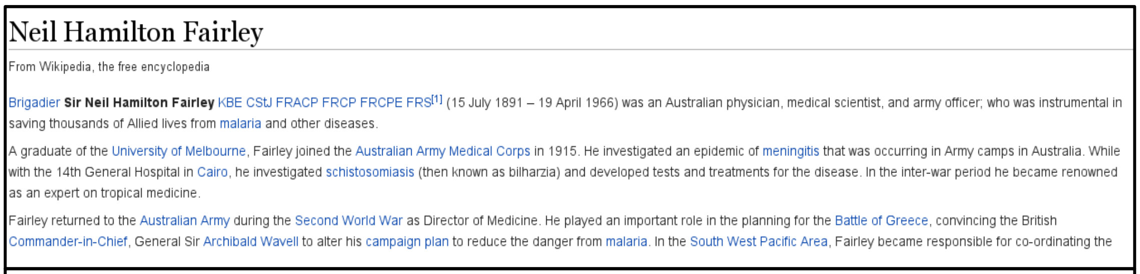
bridge and it is here where the unimodal and cross-modal versions of my projection function differ. This bridge was achieved by simply learning a *second set* of K binary classifiers to predict the *annotation hashcode* bits using the *visual descriptors* as features (Section 6.5.2.3). In this way the 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





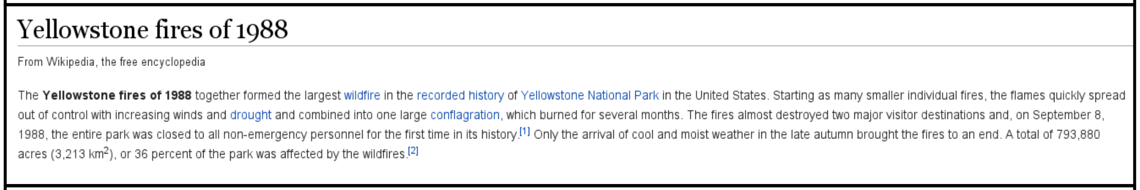
Neil Hamilton Fairley

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.

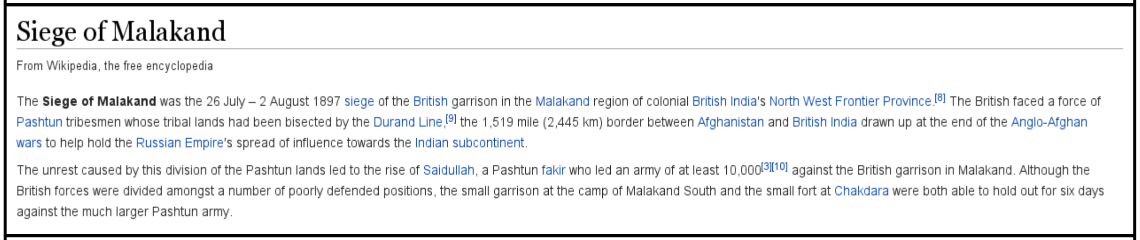
Fairley returned to the Australian Army during the Second World War as Director of Medicine. He played an important role in the planning for the Battle of Greece, convincing the British Commander-in-Chief, General Sir Archibald Wavell to alter his campaign plan to reduce the danger from malaria. In the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

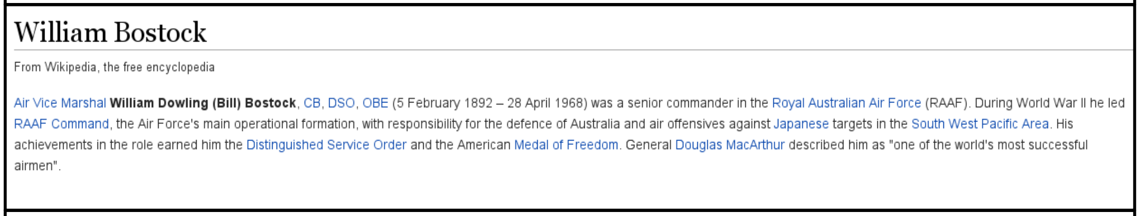


Siege of Malakand

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.

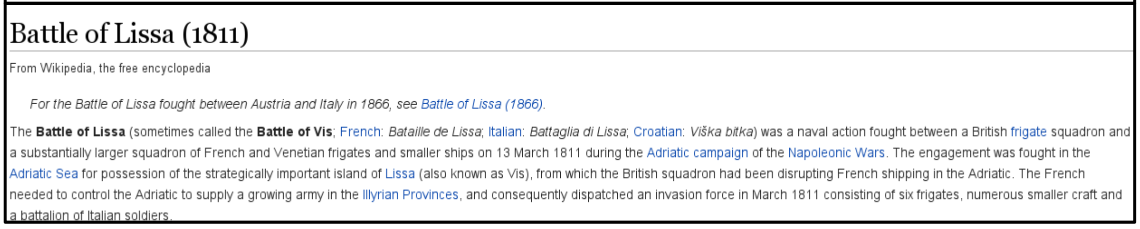
The unrest caused by this division of the Pashtun lands led to the rise of Saidullah, a Pashtun fakir who led an army of at least 10,000[3][10] against the British garrison in Malakand. Although the British forces were divided amongst a number of poorly defended positions, the small garrison at the camp 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".



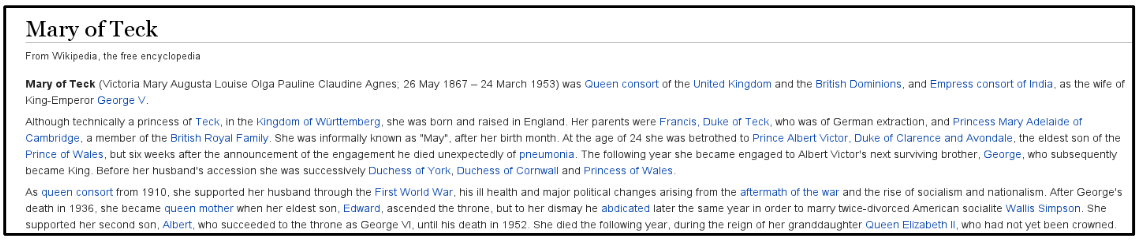
Battle of Lissa (1811)

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).



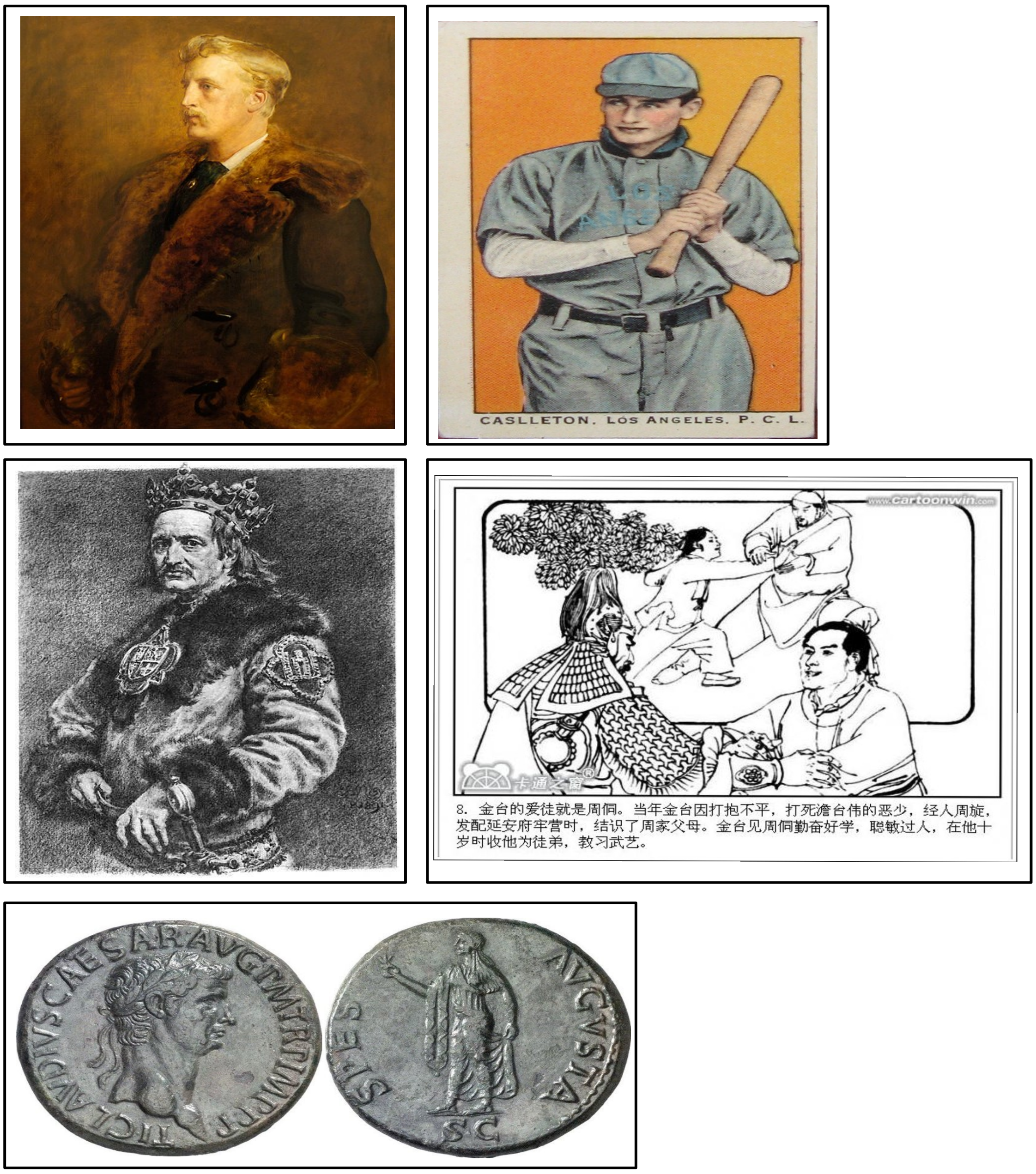
Mary of Teck

From Wikipedia, the free encyclopedia

Mary of Teck (Victoria Mary Augusta Louise Olga Pauline Claudine Agnes; 26 May 1867 – 24 March 1953) was Queen consort of the United Kingdom and the British Dominions, and Empress consort of India, as the wife of King-Emperor George V.

Although technically a princess of Teck, in the Kingdom of Württemberg, she was born and raised in England. Her parents were Francis, Duke of Teck, who was of German extraction, and Princess Mary Adelaide of Cambridge, a member of the British Royal Family. She was informally known as "May", after her birth month. At the age of 24 she was betrothed to Prince Albert Victor, Duke of Clarence and Avondale, the eldest son of the Prince of Wales, but six weeks after the announcement of the engagement he died unexpectedly of pneumonia. The following year she became engaged to Albert Victor's next surviving brother, George, who subsequently became King. Before her husband's accession she was successively Duchess of York, Duchess of Cornwall and Princess of Wales.

As queen consort from 1910, she supported her husband through the First World War, his ill health and major political changes arising from the aftermath of the war and the rise of socialism and nationalism. After George's death in 1936, she became queen mother when her eldest son, Edward, ascended the throne, but to her dismay he abdicated later the same year in order to marry twice-divorced American socialite Wallis Simpson. She supported her second son, Albert, who succeeded to the throne as George VI, until his death in 1952. She died the following year, during the reign of her granddaughter Queen Elizabeth II, who had not yet been crowned.


CASLLETON. LOS ANGELES. P. C. L.
www.cartoonwin.com
卡通之窗
TI CLAVDIVS CAESAR AVG P M TR P IMP P P
SPES AVGVSTA
S C

8. 金台的爱徒就是周侗。当年金台因打抱不平，打死澹台伟的恶少，经人周旋，发配延安府牢营时，结识了周家父母。金台见周侗勤奋好学，聪敏过人，在他十岁时收他为徒弟，教习武艺。

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.

Chapter 7

Learning Hypersurfaces and Quantisation Thresholds

The research presented in this Chapter has been previously published in Moran (2016).

7.1 Introduction

In Chapter 1 I motivated this thesis by arguing how both steps of the hashcode generation pipeline, namely projection and quantisation, were readily amenable to improvement through data-driven learning of the hashing hypersurfaces and quantisation thresholds. In Chapters 4-5 I focused my attention on improving the quantisation step of the hashcode generation pipeline. I found that optimising the position of multiple thresholds per projected dimension was more effective than placing a single threshold by default at zero for mean centered data. In Chapter 6 I improved the projection step by learning hashing hypersurfaces that were adapted to the distribution of the data. In this latter case I found that learnt hypersurfaces fractured the input feature space in a way that resulted in more true nearest neighbours falling within the same partitioned regions compared to a purely random partitioning. In both cases I was encouraged to find statistically significant increases in image retrieval effectiveness compared to existing models that set both parameters independent of the data distribution. In this chapter, I bring together and consolidate the main contributions of this dissertation by performing two experiments. Firstly, in Section 7.2, I combine the multi-threshold quantisation models of Chapters 4-5 with the graph regularised projection function of Chapter 6 to form a hashing model that learns both the hashing hypersurfaces and multiple quantisation thresholds per projected dimension. I measure the performance of the combined model with respect to baselines that learn either parameter in isolation. Secondly, in Section 7.3.3.2, I then appeal to the Computer Vision practitioner by conducting an experiment which seeks to find the most effective combination of the novel models introduced in this thesis for the task of image retrieval.

7.2 Learning Hypersurfaces and Thresholds

7.2.1 Problem Definition

In this chapter I depart from my earlier contributions by learning both the quantisation thresholds and the hashing hypersurfaces *in the same model* so that neighbouring points $\mathbf{x}_i \in \mathbb{R}^D, \mathbf{x}_j \in \mathbb{R}^D$ are more likely to have similar hashcodes $\mathbf{b}_i \in \{-1,1\}^K, \mathbf{b}_j \in \{-1,1\}^K$.

7.2.2 Overview of the Approach

To achieve the learning objective outlined in Section 7.2.1. I take the most straightforward pipeline-based approach in this chapter. Specifically I couple both models by quantising the projections generated by my supervised projection function introduced in Chapter 6 using the multi-threshold quantisation models of Chapters 4-5. The objective is to learn a set of K linear hypersurfaces[1] $\left\{\mathbf{w}_k \in \mathbb{R}^D\right\}_{k=1}^K$ and a set of T quantisation thresholds $\mathbf{t}_k = [t_{k1}, t_{k2}, \ldots, t_{kT}]$ where $t_k \in \mathbb{R}$ and $t_{k1} < t_{k2} \ldots < t_{kT}$ for each of the K projected dimensions $\left\{\mathbf{y}^k \in \mathbb{R}^{N_{trd}}\right\}_{k=1}^K$ produced by those hypersurfaces. The K hyperplanes are learnt using Algorithm 9 which was presented on page 203 in Chapter 6 while the T thresholds are optimised by maximising Equation 4.6 presented on page 122 in Chapter 4. Equation 4.6 is maximised using Evolutionary Algorithms which were found to be the best performing method of stochastic search for multiple threshold optimisation in Chapter 4 Section 4.3.3.3.

More concretely the model is firstly run for M iterations, in which the following three steps (A-C) are applied during each iteration:

- **A) Regularisation:** Regularise the hashcodes using Equation 7.1:

$$\mathbf{B}_m \leftarrow \operatorname{sgn}\left(\alpha\, \mathbf{S}\mathbf{D}^{-1}\mathbf{B}_{m-1} + (1-\alpha)\mathbf{B}_0\right) \tag{7.1}$$

[1] I leave exploration of non-linear hypersurfaces induced by the radial basis function (RBF) kernel to future work.

$\mathbf{B}_m \in \{-1,1\}^{N_{trd} \times K}$ are the hashcodes for the N_{trd} training data-points at iteration m, the bits in B_0 are initialised using any existing projection function such as LSH or ITQ+CCA, $\alpha \in [0,1]$ is a scalar interpolation parameter, $\mathbf{S} \in \{0,1\}^{N_{trd} \times N_{trd}}$ is the adjacency matrix, $\mathbf{D}$ is a diagonal matrix containing the degree of each node in the graph.

- **B) Partitioning:** Solve the following K constrained optimisation problems in Equation 7.2:

$$\begin{aligned} \text{for } k = 1 \ldots K: \quad & \min \; ||\mathbf{w}_k||^2 + C\textstyle\sum_{i=1}^{N_{trd}} \xi_{ik} \\ \text{s.t.} \quad & B_{ik}(\mathbf{w}_k^\intercal \mathbf{x}_i) \geq 1 - \xi_{ik} \qquad \text{for } i = 1 \ldots N_{trd} \end{aligned} \tag{7.2}$$

  $\mathbf{w}_k \in \mathbb{R}^D$ is the hyperplane normal vector, $\xi_{ik} \in \mathbb{R}_+$ 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 the flexibility of margin.

- **C) Prediction:** Solve for the K projected dimensions $\{\mathbf{y}^k \in \mathbb{R}^{N_{trd}}\}_{k=1}^K$ by computing the following N_{trd} dot products in Equation 7.3:

$$y_i^k = \mathbf{w}_k^\intercal \mathbf{x}_i \;\text{ for } i{=}\{1 \ldots N_{trd}\} \text{ and } k{=}\{1 \ldots K\} \tag{7.3}$$

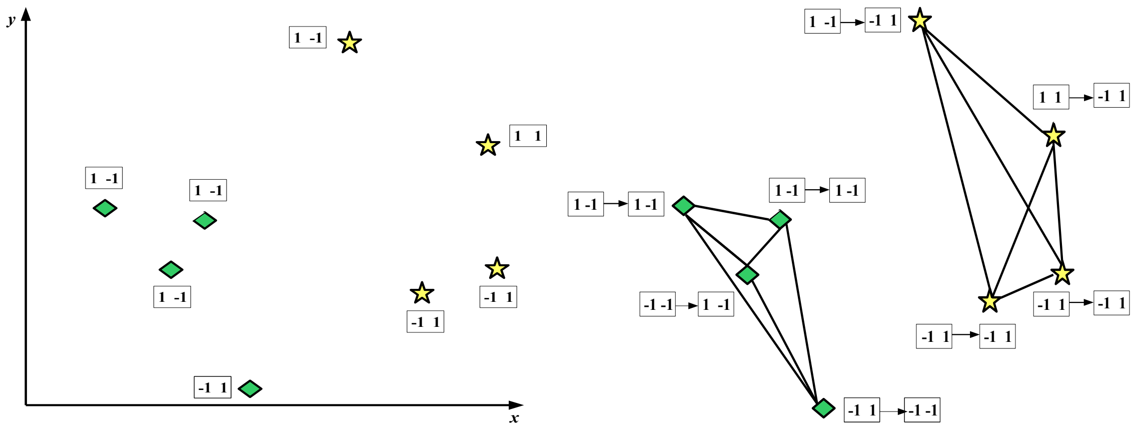
  Standard single bit quantisation (SBQ) is used to generate the updated hashcode matrix $\mathbf{B}_m$. Steps A-C are then repeated for M iterations, which is simply the standard GRH algorithm outlined in Chapter 6, Section 6.2.

Having learnt the projections $\{\mathbf{y}^k \in \mathbb{R}^{N_{trd}}\}_{k=1}^K$ of the N_{trd} data-points in Steps A-C, we then quantise the projections by performing a multi-threshold quantisation using NPQ (Chapter 4) in Step D:

- **D) Quantisation:** Quantise the K projected dimensions $\{\mathbf{y}^k \in \mathbb{R}^{N_{trd}}\}_{k=1}^K$ with thresholds $\mathbf{t}_k = [t_{k1}, t_{k2}, \ldots, t_{kT}]$ learnt by optimising $\mathcal{J}_{npq}(\mathbf{t}_k)$ in Equation 7.4:

$$\mathcal{J}_{npq}(\mathbf{t}_k) = \hat{\alpha} F_1(\mathbf{t}_k) + (1 - \hat{\alpha})(1 - \Omega(\mathbf{t}_k)) \tag{7.4}$$

  where $\hat{\alpha} \in [0,1]$ is a scalar interpolation parameter. The resulting N_{trd} bits for projected dimension k are placed in the *k-th* column of matrix $\mathbf{B}_m$ and the procedure repeated from Step A. Here $T \in [1,3,7,15]$ is the number of thresholds per projected dimension and $F_1(\mathbf{t}_k)$ is a supervised term that leverages the neighbourhood structure encoded in $\mathbf{S}$. More specifically, for a fixed set of



(a) **LSH initialisation** (b) **Regularisation step**

Figure 7.1: In Figure (a) I show a toy 2D space in which I have initialised 2-bit hashcodes for the data-points with LSH. In Figure (b) I show Step A of the model in which the hashcodes are regularised over the adjacency matrix. Lines between points indicate a nearest neighbour relationship. The hashcodes are updated as shown by the directed arrows.

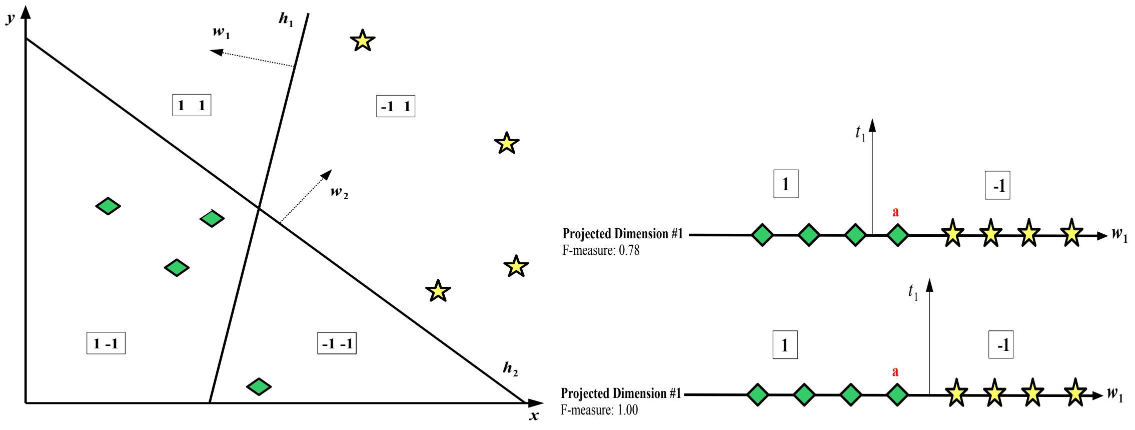
thresholds $\mathbf{t}_k = [t_{k1} \ldots t_{kT}]$, I define a per-projected dimension indicator matrix $\mathbf{P}^k \in \{0,1\}^{N_{trd} \times N_{trd}}$ with the property given in Equation 7.5

$$P^k_{ij} = \begin{cases} 1, & \text{if} \quad \exists_\gamma \quad s.t. \quad t_{k\gamma} \leq (y^k_i, y^k_j) < t_{k(\gamma+1)} \\ 0, & \text{otherwise.} \end{cases} \tag{7.5}$$

where the index $\gamma \in \mathbb{Z}$ spans the range: $0 \leq \gamma \leq T$, and the scalar quantity T denotes the total number of thresholds partitioning a given projected dimension. Intuitively, matrix $\mathbf{P}^k$ indicates whether or not the projections (y^k_i, y^k_j) of any pair of data-points $(\mathbf{x}_i, \mathbf{x}_j)$ fall within the same thresholded region of the projected dimension $\mathbf{y}^k \in \mathbb{R}^{N_{trd}}$. Given a particular instantiation of the thresholds $[t_{k1} \ldots t_{kT}]$, the algorithm counts the number of *true positives* (TP), *false negatives* (FN) and *false positives* (FP) across all regions (Equations 7.6-7.8).

$$TP = \frac{1}{2}\sum_{ij} P_{ij} S_{ij} = \frac{1}{2}\|\mathbf{P} \circ \mathbf{S}\|_1 \tag{7.6}$$

$$FN = \frac{1}{2}\sum_{ij} S_{ij} - TP = \frac{1}{2}\|\mathbf{S}\|_1 - TP \tag{7.7}$$



(a) **Partitioning step** (b) **Quantisation step**

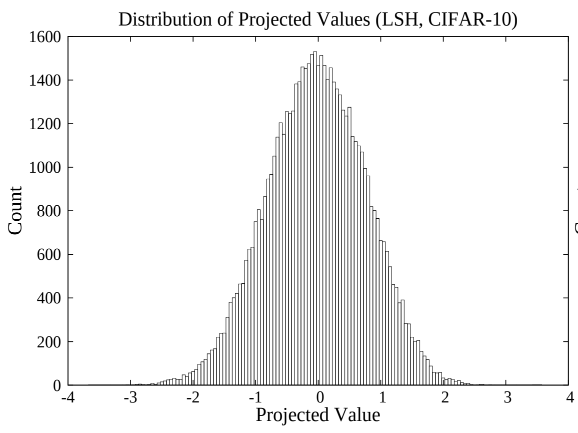
Figure 7.2: In Figure (a) I show the partitioning step (Step B) where linear hypersurfaces are positioned to separate opposing bits (-1,1) with maximum margin. In Figure (b) right I show Step D using the projections for hyperplane $\mathbf{h}_1$ as an example. The top-most diagram shows the quantisation obtained with a threshold placed at zero. Point a is on the wrong side of the threshold. The bottom diagram shows the result obtained by optimising the threshold t_1 to maximise pairwise F_1. In this case the threshold is shifted so that point a receives the same bits as its neighbours. NPQ therefore corrects quantisation boundary errors committed by GRH: this is the crux of the contribution in this chapter.

$$FP = \frac{1}{2}\sum_{ij} P_{ij} - TP = \frac{1}{2}\|\mathbf{P}\|_1 - TP \tag{7.8}$$

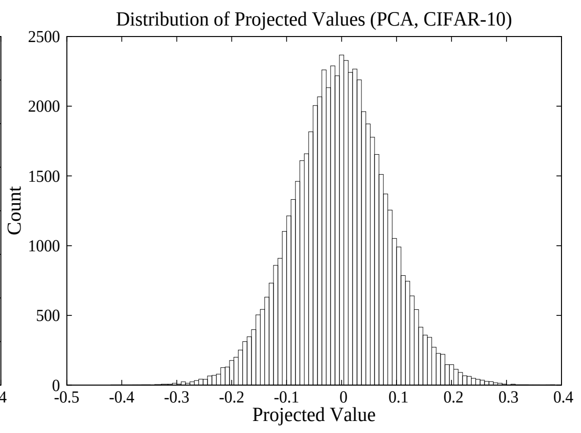
where $\circ$ denotes the Hadamard (elementwise) product and $\|.\|_1$ is the L_1 matrix norm defined as $\|\mathbf{X}\|_1 = \sum_{ij}|X_{ij}|$. Intuitively TP is the number of positive pairs that are found within the same thresholded region, FP is the proportion of negative pairs found within the same region, and FN are the proportions of positive pairs found in different regions. The TP, FP and FN counts are combined using the familiar set-based F_1-measure (Equation 7.9):

$$F_1(\mathbf{t}_k) = \frac{2\|\mathbf{P}\circ\mathbf{S}\|_1}{\|\mathbf{S}\|_1 + \|\mathbf{P}\|_1} \tag{7.9}$$

Intuitively maximisation of Equation 7.9 encourages a clustering of the projected dimension so that as many of the must-link and cannot-link constraints encoded in the adjacency matrix $\mathbf{S}$ are respected. $\mathcal{J}_{npq}$ is the optimisation objective of



(a) **Locality Sensitive Hashing (LSH)**



(b) **Principal Component Analysis (PCA)**

Figure 7.3: Distribution of projected values for two randomly chosen GRH projected dimensions on the CIFAR-10 dataset. In Figure (a) GRH was initialised with LSH hash-codes while in Figure (b) I initialised GRH with hashcodes computed by PCA. In both cases the region of highest projected value density is around zero.

my multi-threshold quantisation algorithm which is defined in more detail in Chapter 4, Section 4.2.2.

In Figures 7.1-7.2, I provide an intuitive overview of the algorithm with an example two-dimensional example problem.

The question arises as to why we might expect learning the hypersurfaces and thresholds within the same model to lead to enhanced retrieval effectiveness. In Figure 7.3 I show projected value histograms created by my graph regularised projection function (GRH). It is clear that GRH projections tend to cluster around zero along each projected dimension. Following a similar argument as for LSH and PCA projections in Chapter 2 Section 2.5.1, I argue that placing a single threshold directly at zero along a GRH projected dimension can be sub-optimal given that it may separate out many true nearest neighbours on opposite sides of the threshold as this is the region of highest point density. In this chapter, I apply the experimental findings of Chapter 4 and hypothesise that optimising the position of multiple thresholds based on the distribution of the data can mitigate this issue with quantisation boundary errors and boost the retrieval effectiveness of GRH. To study this research question I therefore combine GRH with the threshold learning algorithms presented in Chapters 4-5 (NPQ, VBQ) and compare the resulting model to competitive baselines that learn either parameter individually. I present my experimental results in Section 7.3.

7.3 Experimental Evaluation

7.3.1 Experimental Configuration

The experiments in this section are directed towards answering the following single hypothesis:

- H_1: *Learning the hashing hypersurfaces and multiple quantisation thresholds as part of the same model can give a higher retrieval effectiveness than a model which learns either the hypersurfaces or thresholds.*

To answer this hypothesis I use an identical experimental setup to that employed in Chapters 4-5. This experimental configuration is standard in the relevant literature (Kong et al. (2012); Kong and Li (2012a,b); Kulis and Darrell (2009); Raginsky and Lazebnik (2009); Gong and Lazebnik (2011)) and is shown in Table 7.1 for reading convenience. Concretely I define the groundtruth nearest neighbours using the ε-NN paradigm, that is if a data-point is within a radius of ε to the query then it is deemed to be true nearest neighbour for the purposes of evaluation as explained in Chapter 3, Section 3.3.1. Retrieval effectiveness is computed using the standard Hamming ranking evaluation paradigm (Chapter 3, Section 3.4.1) and the area under the precision recall curve (AUPRC) (Chapter 3, Section 3.6.3). Note that I am using a different groundtruth definition (ε-NN) and main evaluation metric (AUPRC) compared to Chapter 6. This means that the quantitative results in this Chapter and Chapter 6 are mostly not directly comparable.

I test the model on the standard CIFAR-10 dataset introduced in Chapter 3, Section 3.2, and leave examination of the performance on other image datasets to future work. To define the test queries ($\mathbf{X}_{teq} \in \mathbb{R}^{N_{teq} \times D}$) I randomly sample $N_{teq} = 1{,}000$ data points with the remaining points forming the database ($\mathbf{X}_{db} \in \mathbb{R}^{N_{db} \times D}$). The precise dataset split is shown in Table 7.2. In all experiments the hypersurfaces and thresholds are learnt on the training dataset ($\mathbf{X}_{trd} \in \mathbb{R}^{N_{trd} \times D}$). These hypersurfaces and thresholds are then used to quantise the test dataset projections ($\mathbf{X}_{teq} \in \mathbb{R}^{N_{teq} \times D}$). The reported AUPRC figures are computed using repeated random sub-sampling cross-validation over ten independent runs. The Wilcoxon signed rank test (Smucker et al. (2007)) is used for measuring statistical significance. The unit of the significance test is a pair of AUPRC values pertaining to the two systems under comparison and resulting from a particular random split of the dataset. In all presented result tables the symbol

Parameter	**Setting**	**Chapter Reference**
Groundtruth Definition	ε-NN	Chapter 3, Section 3.3.1
Evaluation Metric	AUPRC	Chapter 3, Section 3.6.3
Evaluation Paradigm	Hamming Ranking	Chapter 3, Section 3.4.1
Random Partitions	10	Chapter 3, Section 3.5
Number of Bits (K)	16-128	Chapter 2, Section 2.4

Table 7.1: Configuration of the main experimental parameters for the results presented in this chapter.

Partition	**CIFAR-10**
Test queries (N_{teq})	1,000
Validation queries (N_{vaq})	1,000
Validation database (N_{vad})	10,000
Training database (N_{trd})	2,000
Test database (N_{ted})	59,000

Table 7.2: Literature standard splitting strategy partition sizes for the experiments in the chapter. This breakdown is based on the splitting strategy introduced in Chapter 3, Section 3.5.

▲▲/▼▼ indicates a statistically significant increase/decrease with $p < 0.01$, while ▲/▼ indicates a statistically significant increase/decrease with $p < 0.05$.

7.3.2 Parameter Optimisation

Several hyperparameters need to be set for the graph regularised projection function (GRH) and the multiple threshold quantisation model (NPQ). For NPQ I use evolutionary algorithms with the number of individuals H set to 15 and the number of iterations M set to 15 as was found to be optimal in Chapter 4, Section 4.3.3.3. The NPQ interpolation parameter α is set to $\alpha = 1$ and I use three thresholds ($T = 3$) per projected dimension and the Manhattan hashing codebook and hashcode ranking strategy (Chapter 2, Section 2.5.4). For GRH, I set the number of iterations $M \in \mathbb{Z}_+$, the interpolation parameter $\alpha \in [0, 1]$ and the flexibility of margin $C \in \mathbb{R}_+$ for the SVM by closely following the parameter setting strategy outlined in Chapter 6, Section 6.3.2. Specifically the parameters are set on the held-out validation dataset $\mathbf{X}_{vaq}, \mathbf{X}_{vad}$. I set $C = 1$ and perform a grid search over $M \in \{1 \dots 20\}$ and $\alpha \in \{0.1, \dots, 0.9, 1.0\}$, selecting the setting that gives the highest validation AUPRC. To find M, I stop the sweep when the validation dataset AUPRC falls for the first time, and set M to be the number of the penultimate iteration. Finally M and α are then held constant at their optimised values, and $C \in \{0.01, 0.1, 1, 10, 100\}$. I equally weigh both classes (-1 and 1) in the SVM.

7.3.3 Experimental Results

7.3.3.1 Experiment I: Learning Hashing Hypersurfaces and Multiple Quantisation Thresholds

In this experiment I seek to answer hypothesis H_1 as to whether combining the multiple threshold quantisation algorithm from Chapter 4 with the graph regularised projection function developed in Chapter 6 can be more effective than model either algorithm in isolation. To answer this hypothesis I will follow the experimental setup in Chapter 4 and treat the graph regularised projection function (GRH) as a method for *improving* the projections of existing data-dependent and independent projection functions such as LSH and PCA. I will therefore take the hashcodes produced by the existing projection functions of LSH, PCA, ITQ, SH and SKLSH and use those bits as the initialisation point for GRH in the training hashcode matrix $\mathbf{B}_0$[2]. The results for the retrieval experiments on the CIFAR-10 image collection are shown in Table 7.3, Figures 7.4 (a)-(b) and Figures 7.5 (a)-(b).

In Table 7.3 it is clear that, apart from an ITQ initialisation, learning the hypersurfaces and thresholds as part of the same combined model (GRH+NPQ) yields the highest retrieval effectiveness compared to learning the hypersurfaces (GRH+SBQ) or the thresholds (NPQ, VBQ) independently, as I did in Chapter 6 and Chapters 4-5, respectively[3]. For example, for LSH projections GRH+NPQ gains a relative increase in AUPRC of 60% over NPQ and 28% over GRH+SBQ. Furthermore, the combination of GRH+NPQ outperforms an adaptive threshold allocation (VBQ) by a relative margin of 27%. Each of these increases are found to be statistically significant using a Wilcoxon signed rank test (p-value < 0.01). I am encouraged to find that the superior retrieval effectiveness of GRH+NPQ is maintained when the hashcode length is varied between 16-128 bits for both LSH, PCA, SKLSH and SH projections (Figure 7.4 (a)-(b), Figure 7.5 (a)-(b)). Based on these experimental results I confirm my primary

[2]Please refer to Chapter 2 Sections 2.4, 2.6.2 and Section 2.6.3 for an overview of LSH, PCA, ITQ, SH and SKLSH

[3]I compare to the binary integer linear programme (BILP) variant of VBQ (VBQ_{bilp}) in this Chapter.

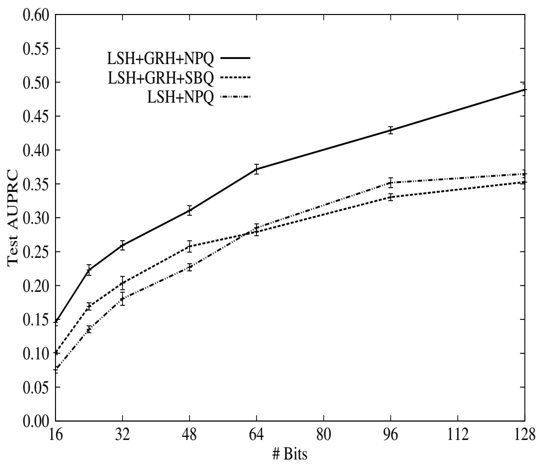
	Model					
	SBQ	**NPQ**	**GRH+SBQ**	**VBQ**	**GRH+NPQ**	**GRH+VBQ**
LSH	0.0954	0.1621	0.2023	0.2035	**0.2593▲▲**	0.2420
ITQ	0.2669	**0.3917▲▲**	0.2558	0.3383	0.3670	0.3185
SH	0.0626	0.1834	0.2147	0.2380	0.2958	**0.3003▲▲**
PCA	0.0387	0.1660	0.2186	0.2579	0.2791	**0.2897▲▲**
SKLSH	0.0513	0.1063	0.1652	0.2122	**0.2566▲▲**	0.2334

Table 7.3: AUPRC on the CIFAR-10 dataset with a hashcode length of 32 bits. The quantisation algorithms listed on the first row are used to quantise the projections from the hash functions in the first column. ▲▲ indicates statistical significance (Wilcoxon signed rank test, $p < 0.01$) when comparing NPQ and GRH+NPQ (or GRH+VBQ).

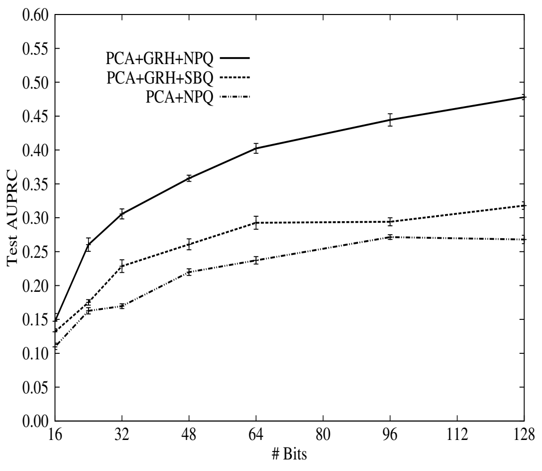
hypothesis (H_1) that learning of the hypersurfaces and thresholds in the same combined model can outperform a model which learns either the thresholds or the hypersurfaces, but not both.

The fact that the retrieval effectiveness for PCA+GRH+NPQ does not tail off in Figure 7.4 (b) as the hashcode length increases is a particularly significant finding. In Chapters 4-5 I observed that PCA projections quantised with *multiple* thresholds per projected dimension tend to approach a plateau in AUPRC with increasing hashcode length. This observation is related to the more unreliable PCA hyperplanes with low eigenvalue which tend to capture very little variance in the input feature space. Using the normal vectors of those hyperplanes to generate hashcode bits is therefore likely to add little to the overall retrieval effectiveness. In contrast, refining PCA hyperplanes with NPQ and GRH appears to entirely overcome this issue, yielding a retrieval effectiveness that continues to increase past a hashcode length of $K = 128$ bits. PCA+GRH+NPQ is therefore an effective means of overcoming the imbalanced variance problem associated with PCA hyperplanes, an issue that has attracted significant research effort in the learning to hash community (Weiss et al. (2008), Kong and Li (2012b), Gong and Lazebnik (2011)).

In Chapter 5 I devised an algorithm (VBQ) that intelligently picked the most discriminative subset of hyperplanes while simultaneously deciding how many thresholds to allocate to those hyperplanes. The pertinent question in this Chapter is whether there is an additive benefit obtainable from refining the PCA, LSH, SH and SKLSH hyperplanes with GRH and then subsequently quantising those GRH projections with VBQ.



(a) **LSH projections**

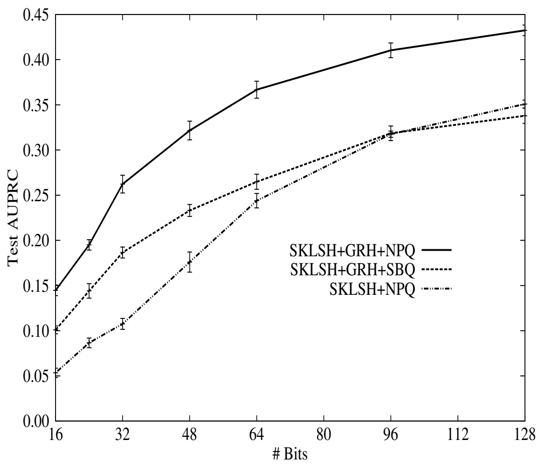


(b) **PCA projections**

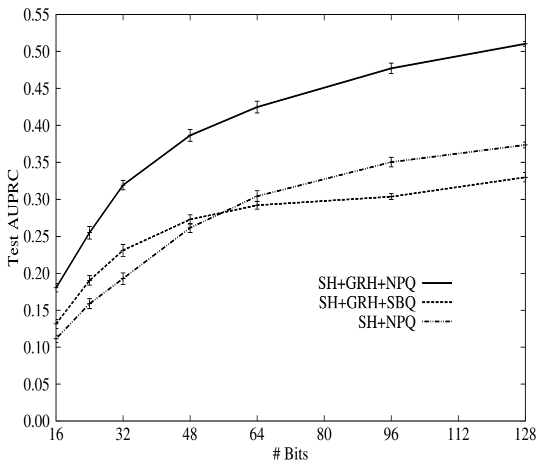
Figure 7.4: Learning the hashing hypersurfaces and quantisation thresholds jointly (GRH+NPQ) versus independently (GRH+SBQ, SBQ) over a haschode length of 16-128 bits. Results are shown for LSH projections (Figure (a)) and PCA projections (Figure (b)).

The results in Table 7.3 confirm that an additive benefit *does* exist for these projections, namely GRH+VBQ has a significantly higher effectiveness than either VBQ or GRH+SBQ alone when quantising PCA, LSH, SH and SKLSH projections. Nevertheless, I also observe the result that for these four projection functions GRH+VBQ *does not* significantly outperform GRH+NPQ. This latter result suggests that both VBQ and NPQ are *equivalently effective* in quantising the GRH refined projections and no further boost in effectiveness is possible by assigning different number of thresholds per projected dimension. The end user can therefore save computation time by using $K/2$ of the K available hyperplanes and quantising all $K/2$ with a uniform three thresholds per projected dimension, rather than rely on VBQ to find a good data-driven allocation.

The retrieval results obtained when ITQ provides the initialisation of the hyperplanes suggest a somewhat different story. Firstly I observe from Table 7.3 that there is no significant difference between the AUPRC from simply quantising ITQ projections directly with single bit quantisation (SBQ) (0.2669) and further refining the hyperplanes with GRH (0.2558). This result therefore strongly suggests that the ITQ hyperplanes are already providing a very good partitioning of the input feature space under the ϵ-NN groundtruth definition, with GRH being unable to better that positioning as it readily does for LSH, SH, PCA and SKLSH projection functions. However quantising the projections from the *GRH refined* ITQ hyperplanes with both NPQ



(a) **SKLSH projections**



(b) **SH projections**

Figure 7.5: Learning the hashing hypersurfaces and quantisation thresholds jointly (GRH+NPQ) versus independently (GRH+SBQ, SBQ) over a haschode length of 16-128 bits. Results are shown for SKLSH projections (Figure (a)) and SH projections (Figure (b)).

(GRH+NPQ) and VBQ (GRH+VBQ) gives significantly lower retrieval effectiveness (0.3670, 0.3185) than simply taking the ITQ hyperplanes directly and quantising their projections with NPQ (0.3917). Using both GRH and NPQ or VBQ in the same model therefore does *not* give an additive benefit in retrieval effectiveness as it does for the LSH, SKLSH, PCA and SH projection functions. This finding suggests that ITQ+NPQ is the best model combination and leads to the highest image retrieval effectiveness for the literature standard groundtruth definition (ε-NN) and Hamming ranking evaluation metric (AUPRC). I explore this result further in Section 7.3.3.2.

7.3.3.2 Experiment II: Finding the Model Combination with Highest Retrieval Effectiveness

In my final experiment of this chapter I will compare the effectiveness of the best combination of models resulting from the novel contributions made in this thesis to the highest performing baselines from the literature. Table 7.4 presents the retrieval results on the CIFAR-10 dataset. I select ITQ+NPQ as my best model which quantises ITQ projections with the multi-threshold quantisation algorithm (NPQ). NPQ is parameterised with $T = 3$ thresholds per projected dimensions and is configured to use the Manhattan Quantisation binary codebook and Manhattan distance hashcode ranking strategy. This combination was found to give the overall highest retrieval effectiveness out of all the model combinations I considered in preliminary experiments. As baselines for comparison I select two of the strongest performing models from the relevant literature. In previous chapters I found Iterative Quantisation (ITQ) (Gong and Lazebnik (2011)) to give high retrieval effectiveness, in addition to the Supervised Hashing with Kernels (KSH) model of Liu et al. (2012). I observe in Table 7.4 that ITQ+NPQ significantly (Wilcoxon signed rank test, $p < 0.01$) outperforms the strong non-linear baseline model of KSH, while also substantially outperforming the other baseline of ITQ on the task of image retrieval. Combining my proposed quantisation model (NPQ) with ITQ therefore leads to a new state-of-the-art retrieval performance on the standard CIFAR-10 image dataset.

	16	**32**	**48**	**64**	**128**
ITQ+NPQ	**0.2439▲▲**	**0.3991▲▲**	**0.4567▲▲**	**0.5019▲▲**	**0.5501▲▲**
KSH	0.1654	0.2878	0.3511	0.3820	0.4397
ITQ	0.1734	0.2638	0.3072	0.3382	0.3646

Table 7.4: The best model combination resulting from research in this thesis ITQ+NPQ compared against the models with the highest retrieval effectiveness from the learning to hash literature (ITQ, KSH). ▲▲ indicates statistical significance (Wilcoxon signed rank test, $p < 0.01$) versus KSH.

7.4 Conclusions

In this chapter I have consolidated the novel contributions made throughout this thesis from two angles. Firstly I conducted an initial investigation into coupling the hypersurface and threshold learning algorithms developed in Chapters 4-6. The projections formed by my graph regularised projection function (GRH) were quantised with my multi-threshold quantisation models (NPQ, VBQ). My findings resulting from these model combinations were:

- Learning the hashing hypersurfaces and quantisation thresholds in the same hashing model (GRH+NPQ) can give a retrieval effectiveness higher than a model which learns either the hashing hypersurfaces (GRH) or the quantisation thresholds (NPQ, VBQ). Quantitative results supporting this claim are presented in Section 7.3.3.1, Table 7.3.

In my experiments I treated the GRH and NPQ models as methods for refining the hypersurfaces of existing projection functions such as LSH or PCA. With this view in mind my experimental evaluation found that the GRH+NPQ model combination was shown to be most beneficial for a broad selection of data-independent and dependent projection functions other than the Iterative Quantisation (ITQ) model of Gong and Lazebnik (2011).

I then consolidated my research from a second angle that involved finding the overall best performing model combination that leveraged one or more of the ideas presented throughout this dissertation. I made the following experimental finding in this regard:

- The model combination ITQ+NPQ significantly outperforms a set of strong baseline models from the learning-to-hash literature yielding state-of-the-art retrieval effectiveness. This claim is supported by the results in Section 7.3.3.2, Table 7.4.

For the Computer Vision practitioner, I would therefore advocate the use of the ITQ+NPQ model combination for image retrieval. I found this model combination to be simple to engineer, fast to run in both training and hashcode prediction time for the relatively low-dimensional image datasets considered in this thesis while also exhibiting state-of-the-art image retrieval effectiveness in my experimental evaluation.

In Chapter 8 I conclude this dissertation by providing a summary of my main research findings, alongside several pointers to potentially fruitful avenues for future work in the field.

Chapter 8

Conclusions

8.1 Introduction

In this chapter I conclude this thesis by summarising the main content of the dissertation in Section 8.2 and discussing my novel contributions and experimental findings in Section 8.3. I present in Section 8.4 several suggestions as to how the research presented in this thesis could be extended in the future, with a focus on directions that I consider to be potentially most fruitful.

8.2 Thesis Summary

In this thesis I explored the benefits of *learning* similarity preserving binary hashcodes for hashing-based approximate nearest neighbour (ANN) search. In Chapters 1-2, I described and motivated the problem of nearest neighbour search which involves finding the most similar data-point(s) to a query in a large database. We saw how this operation is truly fundamental in many diverse fields of study, from constructing noun similarity lists from web-scale textual datasets (Ravichandran et al. (2005)) to the automatic detection of thousands of object classes on a single machine (Dean et al. (2013)). The simplicity of this definition belies the considerable complexity of solving this search problem in a manner that does not require exhaustively comparing the query to every single data-point in the database. I specifically focused on the sub-field of approximate nearest neighbour search in this dissertation which encompasses a class of algorithms that seek to generate *similarity preserving* binary hashcodes for the query and database data-points. These binary hashcodes have the critical property of being more similar - that is sharing more bits in common - for more similar data-points. In Chapter 2 we have seen how we can then obtain an attractive *constant query time* by using these binary hashcodes as the indices into the buckets of a set of hashtables and only returning colliding data-points as candidate nearest neighbours. This vast reduction in the search space is the primary advantage of hashing-based ANN search algorithms. However as I also discussed in Chapter 2 the faster query time comes at the cost of both a non-zero probability of failing to retrieve the closest neighbour in all cases and typically the requirement of relatively long hashcodes and multiple hashtables for an adequate level of precision and recall.

The original research presented in Chapters 4-7 introduced a suite of novel data-driven algorithms for improving the retrieval effectiveness of existing models for hashing-based ANN search at the cost of a one time offline training phase. My premise throughout this thesis was that learning the binary hashcodes by explicitly taking the *distribution of the input data* into consideration could yield much more compact and discriminative hashcodes that would improve search effectiveness over and above the state-of-the-art. In Chapter 2 I identified two key operations used by existing hashing-based ANN search algorithms to generate hashcodes both of which are amenable to data-driven learning: *projection* followed by *quantisation*. The projection step involves fracturing the input feature space into a set of disjoint polytope-shaped regions using a set of randomly drawn hyperplanes, with each region so-formed constituting a hashtable bucket. The hashcode for a data-point is therefore a geometric identifier specifying the unique region within which that data-point resides and can be computed by simply determining the position of the data-point with respect to each of the hyperplanes, with a '0' appended to the hashcode if the data-point is on one side of a hyperplane and a '1' otherwise. Computationally this operation involves two fundamental steps: a dot product (projection) onto the normal vector to each hyperplane followed by a thresholding (quantisation) operation on the resulting projected values.

I argued in Chapter 2 that these two operations - projection and quantisation - are responsible for the overall locality preserving quality of the hashcodes. For example, if a hyperplane happens to partition two true nearest neighbours or if a quantisation threshold separates the two related data-points then the hashcodes for those data-points will ultimately be more dissimilar, sharing less bits in common. In Chapter 2 it was further highlighted how Locality Sensitive Hashing (LSH) - a popular family of randomised algorithms for solving the ANN search problem - positioned the hashing hyperplanes randomly and the quantisation thresholds statically (at zero), *independent of the data distribution*. I hypothesised that the data-independent setting of these two critical parameters negatively impacted the retrieval effectiveness of hashing-based ANN search and I set out in Chapters 4-7 to investigate this claim.

Concretely, in Chapter 4, I introduced a novel algorithm for optimising one or more quantisation thresholds using an objective function that explicitly targeted the number of true nearest neighbours that are assigned the same bits. Rather than a single threshold placed statically at zero along a projected dimension I instead advocated a data-adaptive *optimisation* of *multiple* thresholds. In Chapter 5 this quantisation model was extended to automatically learn the *optimal quantity* of thresholds for each hyperplane based on a novel measure of *hyperplane informativeness*. Hyperplanes that better preserved the neighbourhood relationships between the data-points in the input feature space were rewarded with a greater allocation of thresholds. Locality preserving hyperplanes typically result in projections with more related data-points clumped together, a structure that can be better exploited with a finer quantisation granularity consisting of many thresholds. Having instilled a degree of data-dependence into the quantisation operation I turned my attention to learning the hashing hyperplanes in Chapter 6. I introduced a *three-step iterative algorithm* that utilised a small amount of pairwise supervision to guide the placement of the hashing hyperplanes in a way that attempted to avoid dividing regions of the space dense in true nearest neighbours. This model was further adapted to learn hyperplanes that generated similar hashcodes for similar data-points in *two different modalities*, such as images and text. Chapter 7 consolidated the research in this thesis by showing how the hashing hyperplanes and quantisation thresholds could be learnt as part of the same hashing model, thereby unifying the contributions put forward in Chapters 4-6. In all cases experimental analysis suggested my data-driven algorithms significantly improved retrieval effectiveness over incumbent models that set the hyperplanes or quantisation thresholds independently of the data distribution. I will outline the specific contributions made in each of these four chapters in Section 8.3.

8.3 Contributions and Experimental Findings

In this dissertation I have demonstrated that learning the hashing hyperplanes and quantisation thresholds in a task-specific manner can yield statistically significant improvements in hashing-based approximate nearest neighbour search effectiveness. To investigate this claim I set out to relax the previously ingrained assumptions of existing work regarding how the hashing hyperplanes and quantisation thresholds should be constructed. The following three limiting assumptions were first identified in Chapter 1:

- A_1: Single static threshold placed at zero (for mean centered data)
- A_2: Uniform allocation of thresholds across each dimension
- A_3: Linear hypersurfaces (hyperplanes) positioned randomly

Each of these assumptions led to a new data-driven model that specifically sought to relax that assumption. Furthermore, with each model I empirically set out to test its retrieval effectiveness with respect to the best of prior-art in the field on the primary task of query-by-example image retrieval. I now summarise each model and the findings that arose from their experimental evaluation.

8.3.1 Learning Multiple Quantisation Thresholds

In Chapter 4 I contributed the first known semi-supervised multiple threshold learning algorithm for scalar quantisation in the context of hashing-based ANN search. In most existing work the projections are binarised by placing a single static threshold at zero along each projected dimension. The resulting binary bits are then used to construct the hashcodes for the data-points. My model permitted one or more thresholds to be optimised per projected dimension in addition to the application of any binary codebook to index the quantised regions. Furthermore, my quantisation model was unique in both its objective function and the manner in which this objective function was maximised. I proposed an objective function consisting of an F_1-measure supervised term that was interpolated with an unsupervised term that computed the compactness of the projections along a given dimension. The F_1-measure was computed using an adjacency graph that dictated the pairwise relationships between the data-points in the original feature space. True positives in this case were true nearest neighbour pairs falling into the same quantised regions, false negatives were true nearest neighbours pairs that fell into different regions and false positives were non nearest neighbours that fell into the same quantised regions. Given the non-differentiable nature of this objective function I advocated maximisation by stochastic search (simulated annealing and evolutionary algorithms). My experimental results arising from this research were many and varied and I will only attempt to outline the main findings here. Specifically, I demonstrated that:

- Retrieval effectiveness can always be increased by optimising the quantisation threshold(s) rather than statically placing the threshold at zero along a projected dimension (Chapter 4, Section 4.3.3.5).
- The optimum number of thresholds per projected dimension is projection function specific. For example, Locality Sensitive Hashing (LSH) generally preferred a single threshold while Principal Components Analysis (PCA) hashing benefited from two or more (Chapter 4, Sections 4.3.3.5-4.3.3.6).
- My semi-supervised objective function yielded the best retrieval effectiveness compared to state-of-the-art multi-threshold scalar quantisation models from the literature (Chapter 4, Section 4.3.3.8).

I confirmed that relaxing assumption A_1 is beneficial for retrieval effectiveness in the context of hashing-based approximate nearest neighbour search.

8.3.2 Learning Variable Quantisation Thresholds

In the second contribution of this thesis presented in Chapter 5, I highlighted the importance of learning the appropriate allocation of thresholds per projected dimension. The existing strategy of assigning the same number of thresholds to all projected dimension assumes that the corresponding hyperplanes are of equal locality preserving quality, yet this is frequently not true in real datasets. For example, a PCA hyperplane that captures a large proportion of the variance in the input feature space will tend to be much more discriminative than a hyperplane that captures a much lower proportion of the variance. I argued that the additional structure in the projected dimensions resulting from the more informative hyperplanes should be exploited with a quantisation of a finer granularity using multiple thresholds. This thesis presented, to the best of my knowledge, the first known research that identified and solved this problem in the context of hashing-based ANN search. I advocated the F_β-measure as an original way of quantifying the quality of a hyperplane. This F_β-measure was computed from a data-point adjacency graph with hyperplanes better preserving the neighbourhood structure between the data-points generally obtaining a higher F_β-measure. I introduced two new threshold allocation algorithms that used the computed F_β-measure scores to allocate thresholds to hyperplanes. Both algorithms sought to maximise the cumulative F_β-measure across hyperplanes but did so in two very different ways: one algorithm solved the binary integer linear programme using branch and bound, while the other algorithm used a greedy approach that redistributed thresholds from the least informative to the most informative hyperplanes. The experimental evaluation demonstrated the following main findings:

- F_β-measure is a useful quantity for grading the locality preserving power of a hyperplane (Chapter 5, Section 5.3.3.2).
- Retrieval effectiveness can be increased significantly by learning a variable allocation of quantisation thresholds compared to a uniform allocation of thresholds across projected dimensions (Chapter 5, Section 5.3.3.2).

I demonstrated that relaxing assumption A_2 of existing work leads to significantly higher retrieval effectiveness for hashing-based approximate nearest neighbour search.

8.3.3 Learning the Hashing Hypersurfaces

My third and fourth contributions in Chapter 6 centered around the relaxation of assumption A_3. Existing models for hashing-based ANN search draw the hashing hyperplanes randomly within the input feature space. I contributed a new supervised projection function that instilled a degree of supervision into the placement of the hashing hypersurfaces. My contention was that a small amount of supervision would enable a better positioning of the hashing hypersurfaces in a way that encouraged more true nearest neighbours to fall within the same partitioned regions of the space. The projection function was an iterative three step algorithm reminiscent of the Expectation Maximisation (EM) algorithm. In the first step hashcodes of training data-points were smoothed using a data-point adjacency graph, which had the effect of setting the hashcode for each data-point to be the average of the hashcodes of its nearest neighbours as defined by the adjacency graph. This was my novel method for integrating supervision into the hypersurface learning procedure. In the next step a set of binary classifiers were learnt to predict the regularised bits with maximum margin. This step effectively positioned the hashing hypersurfaces within the space in a way that was consistent with the regularised bits: if two data-points shared a bit in common they were more likely to end up on the same side of the corresponding hypersurface. In the third step of the iterative algorithm the training data-points were re-labelled using the learnt hypersurfaces, which corrected the bits of any data-points that ended up on the wrong side of the hypersurfaces in the previous step. Iterating these three steps for a fixed number of iterations enabled the hypersurfaces to evolve into positions that fractured the input feature space in a manner consistent with the supervisory signal. In my unimodal image retrieval evaluation, I made the following main experimental findings:

- Learnt hashing hyperplanes lead to significantly higher nearest neighbour retrieval effectiveness compared to hyperplanes that are placed randomly in the input feature space (Chapter 6, Section 6.3.3.4).
- Regularising hashcodes over a data-point adjacency graph is a more effective method of integrating supervision into the process of hyperplane learning than a Laplacian Eigenmap dimensionality reduction (Chapter 6, Section 6.3.3.6).
- 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) (Chapter 6, Section 6.3.3.8).
- The training and prediction (hashcode generation) time of the linear variant of my projection function was a fraction of the training time of competitive baseline models (Chapter 6, Section 6.5).
- My supervised projection function attained state-of-the-art retrieval effectiveness on standard image datasets, outperforming a large number of competitive data-dependent and independent hashing models from the literature (Chapter 6, Section 6.3.3.9).

The benefit of relaxing assumption A_3 was confirmed in the context of unimodal image retrieval in which the query and database are of the same feature type (e.g. SIFT features). I further extended this supervised projection function to learn hyperplanes that assigned similar hashcodes to similar data-points in two different modalities, such as text (e.g. TF-IDF vectors) and images. The extension to cross-modal hypersurface learning was surprisingly straightforward: I simply learnt another set of binary classifiers in the image feature space using the hashcodes of the textual data-points as targets. This had the desired effect of making the hypersurfaces in the visual feature space consistent, that is capable of assigning the same bits to similar data-points, with those in the textual feature space. Despite this simplicity I found the following encouraging results:

- Extending the three-step iterative hypersurface learning algorithm to cross-modal hashing yielded state-of-the-art retrieval effectiveness on standard cross-modal datasets, outperforming a large selection of existing models in the field (Chapter 6, Section 6.6).

- Regularising hashcodes over a data-point adjacency graph is more effective for learning cross-modal hypersurfaces than solving an eigenvalue problem to obtain the hypersurfaces (Chapter 6, Section 6.6).
- The training time of my cross-modal projection function was a fraction of the time required by competitive baselines while having a similar prediction (hashcode generation) time (Chapter 6, Table 6.27).

Relaxing assumption A_3 was therefore also found to be beneficial for cross-modal retrieval effectiveness.

8.3.4 Learning Hypersurfaces and Quantisation Thresholds

In the final contribution of this thesis I conducted a preliminary exploration into the effect on retrieval effectiveness of learning both the hashing hypersurfaces and multiple quantisation thresholds jointly. This research presented in Chapter 7 combined the multiple threshold quantisation algorithms introduced in Chapters 4-5 with the iterative hypersurface learning algorithm presented in Chapter 6. In doing so I created a fully data-adaptive hashing pipeline of projection followed by quantisation. To connect both models I binarised the low-dimensional projections computed by the hypersurface learning algorithm using the multiple threshold quantisation algorithm. On the standard task of query-by-example image retrieval I made the following encouraging finding:

- Learning the hashing hypersurfaces and the quantisation thresholds as part of the same hashing model gives a retrieval effectiveness significantly greater than learning either parameter individually (Chapter 7, Section 7.3.3.1).

8.4 Avenues for Future Work

The novel contributions presented in this thesis have but only scratched the surface of this important and flourishing field of research and the potential scope for future research is both many and varied. I will attempt to highlight several potential future directions that I consider particularly promising in this last section.

8.4.1 Groundtruth and Evaluation Metric Correlation with Human Judgments

There has been little previous work that examines the extent to which the evaluation metrics and groundtruth used in the learning to hash field are sensible for learning hashcodes that correlate well with user search satisfaction. For example, ideally it should be the case that a significant increase in the area under the precision recall curve (AUPRC) should also lead to a significant increase in user satisfaction with the retrieved images or documents. Furthermore, in Chapter 3 I introduced the class-based and ε-NN based groundtruth definitions that were subsequently used to evaluate my models in Chapter 4-7. Many datasets of interest do not have manually assigned class labels, and so it would be useful to conduct a user-study as to how metric definitions of nearest neighbour groundtruth, such as the ε-NN groundtruth paradigm outlined in Chapter 3 Section 3.3.1, align with human judgements of item-item similarity. Ideally we would want many related data-points to a given query, as judged by a user, to be contained within the same ε-ball. For the class-based groundtruth used in Chapter 6 and outlined in Chapter 3 Section 3.3.2, this is less of an issue because those labels have been specifically assigned to the images by humans. The outcome of this user study would be expected to inform future developments in the evaluation procedures for hashing-based ANN search algorithms, and would be a valuable contribution to the community.

8.4.2 Online Learning of the Hashing Hypersurfaces

In Chapter 6 the hashing hypersurfaces were constructed in a batch fashion that assumed the entire training dataset would be immediately available for learning. As soon as the hypersurfaces were learnt they were never updated. This batch learning assumption is flawed when we consider many modern data sources of prime interest such as social media streams (e.g. Twitter). Twitter posts, for example, can be modelled as a never-ending, effectively infinite stream of data that could never be inspected in its entirety in a batch fashion (Petrović et al. (2010)). Furthermore streaming data sources are highly likely to exhibit a drift in the distribution of the data over time as, for example, new topics are discussed and the vocabulary changes. Simply learning a set of hypersurfaces once with no possibility of further updates would be an entirely suboptimal approach in this situation. It would be particularly interesting to adapt the hypersurface learning algorithm presented in Chapter 6 to the streaming data scenario in which the hypersurfaces are capable of being updated in an online manner after each labelled pair of data-points are encountered in the stream. To achieve this goal one could potentially investigate the effectiveness of using passive aggressive (PA) classifiers (Crammer et al. (2006)) in place of the support vector machines (SVMs) used in this thesis. The PA classifier is particularly amenable to online learning and would make an ideal starting point for future research on this topic. To the best of my knowledge no online supervised projection function has so far been proposed for application to large-scale streaming data sources. I believe such a model would have significant potential impact in the field. An interesting challenge in this context would be how to efficiently update the hashcodes of existing data-points in the face of changing data. Furthermore, implementation of this model would address a second criticism of the work presented in this thesis, namely the application of the algorithms to datasets of medium size (1 million data-points or less) and of relatively low dimensionality ($D \leq 512$).

8.4.3 Hashing Documents Written in Two Different Languages

The cross-modal extension to my supervised projection function was only tested on images and textual data in this thesis, both of which were represented as low dimensional feature vectors. A particularly interesting extension to the work would involve exploring how the model could be adapted to hash *cross-lingual documents*, for example English and Spanish Wikipedia articles. In this task my goal would be to cluster related cross-lingual documents in the same hashtable buckets, without using any form of machine translation. In contrast to the image and text features used in this thesis multi-lingual document data sources are likely to be very high dimensional when encoded as TF-IDF vectors. The large freely available *parallel and comparable corpora*[1] consisting of similar documents written in different languages would provide the needed pairwise supervision for learning the hashing hypersurfaces, negating any tedious manual effort to obtain the required labels. The cross-lingual projection function could be directly compared and evaluated against Ture et al. (2011), a solution based on machine translation and traditional unimodal Locality Sensitive Hashing (LSH). Given the significant gains in retrieval effectiveness for the cross-modal experiments conducted in this thesis I have strong reason to suspect that cross-lingual hashing with a suitable adaptation of my graph regularised projection function would attract similar

[1] `http://www.statmt.org/europarl/`

gains in performance.

Given that more and more data on the Web is written in different languages I also foresee an online version of this cross-lingual projection function being particularly exciting future work. For example, a fast steaming algorithm for clustering similar tweets written in many different languages into the same hashtable buckets could prove useful to analysts in the financial industry or to linguists interested in studying the linguistic properties of Twitter and other related micro-blogs (Zanzotto et al. (2011)).

8.4.4 Dependent Hypersurfaces and Quantisation Thresholds

The multiple threshold quantisation models introduced in Chapters 4-5 positioned the quantisation thresholds *independently* across each projected dimension. In other words, the learning of the quantisation thresholds for one projected dimension was independent of the learning of the quantisation thresholds for another projected dimension. Inspired by the body of research into *multivariate discretisation* (Bay (2001)) a potential future avenue of research could examine the benefits of inducing a degree of *dependence* between the quantisation thresholds across projected dimensions. A particularly simple, albeit contrived example of a dataset that would not be quantised correctly by independently optimised thresholds is the two dimensional XOR dataset (Bay (2000)). In this case the quantisation algorithm would need to account for the correlation between the different feature dimensions in order to find the optimal positioning of the thresholds.

In a similar vein of research, the supervised projection function introduced in Chapter 6 constructed each hypersurface independently in a simple sequential fashion. Inducing a degree of dependence between the learning of the hypersurfaces might contribute to a reduced redundancy between bits while also permitting hypersurfaces learnt later in the sequence to focus on data-point pairs incorrectly classified by hypersurfaces learnt earlier in the sequence. A straightforward starting point would be to assign a weight to each pair in the adjacency matrix in a similar manner to the Adaboost algorithm (Schapire and Freund (2012)). True nearest neighbours assigned the same bits by earlier hypersurfaces could have their weight decreased while non-nearest neighbours assigned the same bits could have their weights increased. In this way the learning of the hashing hypersurfaces could be gradually biased to focus on data-points pairs that are more difficult to classify, potentially resulting in enhanced retrieval effectiveness.

8.4.5 Closer Integration of the Projection and Quantisation Operations

In Chapter 7, I demonstrated that combining projection function and quantisation threshold learning as part of the same hashing model can lead to significantly better retrieval effectiveness as compared to learning either in isolation. My approach involved a simple concatenation of my quantisation and projection models developed in Chapters 4-6. In effect, the hyperplanes were optimised first and then the quantisation of the projections were optimised during the second step of the two-stage pipeline, with both steps being performed independently, and without knowledge of the other. In future work it would be interesting to explore a combined objective function that both learns the optimal positioning of the hyperplanes while also simultaneously minimising the quantisation loss. This objective could be optimised in a single procedure, for example, by using existing gradient-based optimisers, to exploit synergies between the projection and quantisation operations to mutually influence and reinforce each other. Indeed, there has been recent evidence provided by Zhu et al. (2016) that such a tight coupling of projection and quantisation can lead to significantly better retrieval effectiveness over the standard image datasets considered in this dissertation.

8.5 Concluding Remarks

This thesis has explored the benefits of learning binary hashcodes for fast nearest neighbour search over large-scale datasets, with a particular focus on images. The experimental results have overwhelmingly indicated that significant increases in retrieval effectiveness can be obtained through data-aware hashcodes compared to their data-oblivious counterparts frequently employed in both industry and academia. I hope that the research presented in this dissertation contributes in some small way to the development of increasingly more effective and efficient algorithms for nearest neighbour search.

Appendix A

Definition of Mathematical Notation

Notation	Definition
N	Number of data-points in dataset
D	Dimensionality of data-point feature representation
K	Number of hashcode bits
L	Number of hashtables
Q	Number of query data-points
B	Number bits per projected dimension
T	Number of thresholds per projected dimension
M	Iterations
C	Randomly sampled data-points *or* cluster centroids
$\mathbf{X} \in \mathbb{R}^{N \times D}$	Dataset of N data-points, dimensionality D
$\mathbf{x}_r \in \mathbb{R}^D : \mathbf{x}_r = \mathbf{X}_{r\bullet}$	r^{th} row of matrix $\mathbf{X}$
$\mathbf{x}^c \in \mathbb{R}^N : \mathbf{x}^c = \mathbf{X}_{\bullet c}$	c^{th} column of matrix $\mathbf{X}$
X_{ij}	Element of matrix $\mathbf{X}$ in row i column j
$\mathbf{q} \in \mathbb{R}^D$	Query data-point
$\mathbf{p} \in \mathbb{R}^D$	Arbitrary database data-point
$\mathbf{Y} \in \mathbb{R}^{N \times K}$	Projection matrix of N data-points, dimensionality K
$\mathbf{y}_r \in \mathbb{R}^K : \mathbf{y}_r = \mathbf{Y}_{r\bullet}$	Projected values for r^{th} data-point
$\mathbf{y}^c \in \mathbb{R}^N : \mathbf{y}^c = \mathbf{Y}_{\bullet c}$	c^{th} projected dimension
$d : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$	Distance function e.g. Euclidean distance
$q_k : \mathbb{R} \to \{0,1\}^B$	Quantisation function
$\mathbf{D} \in \mathbb{R}^{N \times N}$	Matrix of data-point distances
$\mathbf{B} \in \{-1,1\}^{N \times K}$	Hashcodes of N data-points each of length K bits
$\mathbf{b}_r \in \{-1,1\}^K : \mathbf{b}_r = \mathbf{B}_{r\bullet}$	Hashcode of r^{th} data-point $\mathbf{x}_r$

$h_k : \mathbb{R}^D \rightarrow \{0,1\}$	Hash function
$g_l : \mathbb{R}^D \rightarrow \{0,1\}^K$	Hash function concatenation $[h_1(.), h_2(.), \ldots, h_K(.)]$
$\mathbf{S} \in \mathbb{R}^{N \times N}$	$S_{ij} = 1$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ are nearest neighbours, 0 otherwise
$\mathbf{h}_k \in \mathbb{R}^D$	Hyperplane
$\mathbf{w}_k \in \mathbb{R}^D$	Hyperplane normal vector
$\mathbf{W} \in \mathbb{R}^{D \times K}$	Matrix of K hyperplane normal vectors
$t_k \in \mathbb{R}$	Scalar threshold
$\mathbf{T} \in \mathbb{R}^{K \times T}$	Matrix of thresholds for each projected dimension
$\mathbf{t}_r \in \mathbb{R}^T : \mathbf{t}_r = \mathbf{T}_{r\bullet}$	Set of thresholds for r^{th} projected dimension
$\kappa : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$	Kernel function
$\gamma \in \mathbb{R}$	Kernel bandwidth parameter
$\lVert\mathbf{X}\rVert_F^2 = \sum_{ij}^N \lvert X_{ij}\rvert^2$	Frobenius L_2 norm of matrix
$\lVert\mathbf{X}\rVert_F^1 = \sum_{ij}^N \lvert X_{ij}\rvert$	Frobenius L_1 norm of matrix
$\mathbf{X} = diag(\mathbf{x})$	Places elements of vector $\mathbf{x}$ on diagonal of matrix $\mathbf{X}$
$sgn(a) \in \{-1,1\}$	Sign function returning 1 for $a > 0$, and -1 otherwise
$[a]_+$	Equal to a if $a \geq 0$, and 0 otherwise

Table A.1: Definition of the mathematical notation used throughout the thesis

Bibliography

Aggarwal, C. C. and Reddy, C. K., editors (2014). *Data Clustering: Algorithms and Applications*. CRC Press.

Albakour, M.-D., Macdonald, C., and Ounis, I. (2015). Using sensor metadata streams to identify topics of local events in the city. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 711–714, New York, NY, USA. ACM.

Andoni, A. and Indyk, P. (2008). Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Commun. ACM*, 51(1):117–122.

Arun, K. S., Huang, T. S., and Blostein, S. D. (1987). Least-Squares Fitting of Two 3-D Point Sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):698–700.

Baluja, S. and Covell, M. (2008). Learning to Hash: Forgiving Hash Functions and Applications. *Data Min. Knowl. Discov.*, 17(3):402–430.

Bawa, M., Condie, T., and Ganesan, P. (2005). LSH Forest: Self-tuning Indexes for Similarity Search. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 651–660, New York, NY, USA. ACM.

Bay, S. D. (2000). Multivariate Discretization of Continuous Variables for Set Mining. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 315–319, New York, NY, USA. ACM.

Bay, S. D. (2001). Multivariate Discretization for Set Mining. *Knowledge and Information Systems*, 3(4):491–512.

Belkin, M. and Niyogi, P. (2003). Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Comput.*, 15(6):1373–1396.

Bengio, Y., Paiement, J.-F., Vincent, P., Delalleau, O., Roux, N. L., and Ouimet, M. (2004). Out-of-Sample Extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA.

Bentley, J. L. (1975). Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, 18(9):509–517.

Berchtold, S., Böhm, C., Braunmüller, B., Keim, D. A., and Kriegel, H.-P. (1997). Fast Parallel Similarity Search in Multimedia Databases. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 1–12, New York, NY, USA. ACM.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, 3:993–1022.

Bohr, H. and Brunak, S. (1989). A Travelling Salesman Approach to Protein Conformation. *Complex Syst.*, 3:9–28.

Brassard, G. and Bratley, P. (1996). *Fundamentals of Algorithmics*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Broder, A. (1997). On the Resemblance and Containment of Documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pages 21–, Washington, DC, USA. IEEE Computer Society.

Bronstein, M. M., Bronstein, A. M., Michel, F., and Paragios, N. (2010). Data fusion through cross-modality metric learning using similarity-sensitive hashing. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 0:3594–3601.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27.

Charikar, M. S. (2002). Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 380–388, New York, NY, USA. ACM.

Chawla, N. V. (2005). Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 853–867. Springer US.

Chua, T.-S., Tang, J., Hong, R., Li, H., Luo, Z., and Zheng, Y.-T. (2009). NUS-WIDE: A Real-World Web Image Database from National University of Singapore. In *Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*, Santorini, Greece.

Chum, O., Philbin, J., and Zisserman, A. (2008). Near Duplicate Image Detection: min-Hash and tf-idf Weighting. In *Proceedings of the British Machine Vision Conference 2008, Leeds, September 2008*, pages 1–10.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online Passive-Aggressive Algorithms. *J. Mach. Learn. Res.*, 7:551–585.

Dantzig, G. B. (1957). Discrete-Variable Extremum Problems. *Operations Research*, 5(2):266–277.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. (2004). Locality-sensitive Hashing Scheme Based on P-stable Distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pages 253–262, New York, NY, USA. ACM.

Dean, T., Ruzon, M., Segal, M., Shlens, J., Vijayanarasimhan, S., and Yagnik, J. (2013). Fast, Accurate Detection of 100,000 Object Classes on a Single Machine. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA.

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society, Series B*, 39(1):1–38.

Diaz, F. (2007). Regularizing Query-based Retrieval Scores. *Inf. Retr.*, 10(6):531–562.

Doersch, C., Singh, S., Gupta, A., Sivic, J., and Efros, A. A. (2012). What Makes Paris Look Like Paris? *ACM Trans. Graph.*, 31(4):101:1–101:9.

Dougherty, J., Kohavi, R., and Sahami, M. (1995). Supervised and Unsupervised Discretization of Continuous Features. In *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*, pages 194–202.

Fan, R., Chang, K., Hsieh, C., Wang, X., and Lin, C. (2008). LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9:1871–1874.

Fayyad, U. M. and Irani, K. B. (1993). Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993*, pages 1022–1029.

Feng, J. (2012). Mobile Product Search with Bag of Hash Bits and Boundary Reranking. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 3005–3012, Washington, DC, USA. IEEE Computer Society.

Finkel, R. A. and Bentley, J. L. (1974). Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Inf.*, 4:1–9.

Freitas, A. A. (2002). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Freund, Y. and Schapire, R. E. (1997). A Decision-theoretic Generalization of On-line Learning and an Application to Boosting. *J. Comput. Syst. Sci.*, 55(1):119–139.

Garcia, S., Luengo, J., Saez, J. A., Lopez, V., and Herrera, F. (2013). A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning. *IEEE Trans. on Knowl. and Data Eng.*, 25(4):734–750.

Goemans, M. X. and Williamson, D. P. (1995). Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. ACM*, 42(6):1115–1145.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA.

Gong, Y. and Lazebnik, S. (2011). Iterative Quantization: A Procrustean Approach to Learning Binary Codes. In *Proceedings of the 2011 IEEE Conference on Computer*

Vision and Pattern Recognition, CVPR '11, pages 817–824, Washington, DC, USA. IEEE Computer Society.

Grauman, K. and Darrell, T. (2004). Fast Contour Matching Using Approximate Earth Mover's Distance. In *CVPR (1)*, pages 220–227.

Grauman, K. and Darrell, T. (2007). Pyramid Match Hashing: Sub-Linear Time Indexing Over Partial Correspondences. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*.

Grauman, K. and Fergus, R. (2013). Learning Binary Hash Codes for Large-Scale Image Search. In Cipolla, R., Battiato, S., and Farinella, G. M., editors, *Machine Learning for Computer Vision*, volume 411 of *Studies in Computational Intelligence*, pages 49–87. Springer Berlin Heidelberg.

Gray, F. (1953). Pulse code communication. US Patent 2,632,058.

Gray, R. M. and Neuhoff, D. L. (2006). Quantization. *IEEE Trans. Inf. Theor.*, 44(6):2325–2383.

Hanson, R. J. and Norris, M. J. (1981). Analysis of Measurements Based on the Singular Value Decomposition. *SIAM Journal on Scientific and Statistical Computing*, 2(3):363–373.

Hardoon, D. R., Szedmak, S., and Shawe-Taylor, J. (2003). Canonical correlation analysis; An overview with application to learning methods. Technical report, Royal Holloway, University of London.

He, K., Wen, F., and Sun, J. (2013). K-Means Hashing: An Affinity-Preserving Quantization Method for Learning Binary Compact Codes. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pages 2938–2945, Washington, DC, USA. IEEE Computer Society.

He, X., Cai, D., and Niyogi, P. (2005). Laplacian Score for Feature Selection. In *Advances in Neural Information Processing Systems 18*, pages 507–514.

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *J. Educ. Psych.*, 24.

Indyk, P. and Motwani, R. (1998). Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA. ACM.

Ingber, L. (1993). Simulated annealing: Practice versus theory. *Mathl. Comput. Modelling*, 18:29–57.

Jegou, H., Douze, M., and Schmid, C. (2011). Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128.

Joachims, T. (2006). Training Linear SVMs in Linear Time. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 217–226, New York, NY, USA. ACM.

Johnson, W. and Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. In *Contemp Math 26*.

Katayama, N. and Satoh, S. (1997). The SR-tree: An Index Structure for High-dimensional Nearest Neighbor Queries. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 369–380, New York, NY, USA. ACM.

Kerber, R. (1992). ChiMerge: Discretization of Numeric Attributes. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pages 123–128. AAAI Press.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Kokiopoulou, E., Chen, J., and Saad, Y. (2011). Trace optimization and eigenproblems in dimension reduction methods. *Numerical Lin. Alg. with Applic.*, 18(3):565–602.

Kong, W. and Li, W. (2012a). Double-Bit Quantization for Hashing. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*

Kong, W. and Li, W. (2012b). Isotropic Hashing. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1655–1663.

Kong, W., Li, W.-J., and Guo, M. (2012). Manhattan Hashing for Large-scale Image Retrieval. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 45–54, New York, NY, USA. ACM.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*.

Kulis, B. (2013). Metric Learning: A Survey. *Foundations and Trends in Machine Learning*, 5(4):287–364.

Kulis, B. and Darrell, T. (2009). Learning to Hash with Binary Reconstructive Embeddings. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 1042–1050.

Kulis, B. and Grauman, K. (2009). Kernelized locality-sensitive hashing for scalable image search. In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*, pages 2130–2137.

Kumar, S. and Udupa, R. (2011). Learning Hash Functions for Cross-view Similarity Search. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI '11, pages 1360–1365. AAAI Press.

Land, A. H. and Doig, A. G. (1960). An Automatic Method of Solving Discrete Programming Problems. *Econometrica*, 28(3):pp. 497–520.

Lawler, E. L. and Wood, D. E. (1966). Branch-And-Bound Methods: A Survey. *Operations Research*, 14(4):pp. 699–719.

Liu, W., Mu, C., Kumar, S., and Chang, S. (2014). Discrete Graph Hashing. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3419–3427.

Liu, W., Wang, J., Ji, R., Jiang, Y., and Chang, S. (2012). Supervised hashing with kernels. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 2074–2081.

Liu, W., Wang, J., Kumar, S., and Chang, S. (2011). Hashing with Graphs. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 1–8.

Liu, X., He, J., and Lang, B. (2013). Reciprocal Hash Tables for Nearest Neighbor Search. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA.*

Lloyd, S. (1982). Least Square Quantization in PCM. *IEEE Trans. Inform. Theory*, 28:129–137.

Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

Martello, S. and Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA.

Matei, B., Shan, Y., Sawhney, H. S., Tan, Y., Kumar, R., Huber, D., and Hebert, M. (2006). Rapid Object Indexing Using Locality Sensitive Hashing and Joint 3D-Signature Space Estimation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(7):1111–1126.

Mehta, S., Parthasarathy, S., and Yang, H. (2005). Toward Unsupervised Correlation Preserving Discretization. *IEEE Trans. on Knowl. and Data Eng.*, 17(9):1174–1185.

Minsky, M. and Papert, S. (1969). *Perceptrons*. MIT Press, Cambridge, MA.

Moran, S. (2016). Learning to Project and Binarise for Hashing-Based Approximate Nearest Neighbour Search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16.

Moran, S. and Lavrenko, V. (2014). Sparse Kernel Learning for Image Annotation. In *Proceedings of International Conference on Multimedia Retrieval*, ICMR '14, pages 113:113–113:120, New York, NY, USA. ACM.

Moran, S. and Lavrenko, V. (2015a). Graph Regularised Hashing. In *Advances in Information Retrieval - 37th European Conference on IR Research, ECIR 2015, Vienna, Austria, March 29 - April 2, 2015. Proceedings*, pages 135–146.

Moran, S. and Lavrenko, V. (2015b). Regularised Cross-Modal Hashing. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 907–910. ACM.

Moran, S., Lavrenko, V., and Osborne, M. (2013a). Neighbourhood Preserving Quantisation for LSH. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 1009–1012, New York, NY, USA. ACM.

Moran, S., Lavrenko, V., and Osborne, M. (2013b). Variable Bit Quantisation for LSH. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 2: Short Papers*, pages 753–758.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.

Oliva, A. and Torralba, A. (2001). Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope. *Int. J. Comput. Vision*, 42(3):145–175.

Osborne, M., Moran, S., McCreadie, R., von Lünen, A., Sykora, M. D., Cano, A. E., Ireson, N., Macdonald, C., Ounis, I., He, Y., Jackson, T., Ciravegna, F., and O'Brien, A. (2014). Real-Time Detection, Tracking, and Monitoring of Automatically Discovered Events in Social Media. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*, pages 37–42.

Petrovic, S. (2012). *Real-Time Event Detection in Massive Streams*. PhD thesis, University of Edinburgh.

Petrović, S., Osborne, M., and Lavrenko, V. (2010). Streaming First Story Detection with Application to Twitter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 181–189, Stroudsburg, PA, USA. Association for Computational Linguistics.

Raginsky, M. and Lazebnik, S. (2009). Locality-sensitive binary codes from shift-invariant kernels. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of* *a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 1509–1517.

Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 1177–1184.

Rajaraman, A. and Ullman, J. D. (2011). *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA.

Rasiwasia, N., Costa Pereira, J., Coviello, E., Doyle, G., Lanckriet, G., Levy, R., and Vasconcelos, N. (2010). A New Approach to Cross-Modal Multimedia Retrieval. In *ACM International Conference on Multimedia*, pages 251–260.

Rasiwasia, N., Mahajan, D., Mahadevan, V., and Aggarwal, G. (2014). Cluster Canonical Correlation Analysis. In *Proceedings of International Conference on Artificial Intelligence and Statistics*.

Rastegari, M., Choi, J., Fakhraei, S., III, H. D., and Davis, L. S. (2013). Predictable dual-view hashing. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1328–1336.

Ravichandran, D., Pantel, P., and Hovy, E. (2005). Randomized Algorithms and NLP: Using Locality Sensitive Hash Function for High Speed Noun Clustering. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 622–629, Stroudsburg, PA, USA. Association for Computational Linguistics.

Rijsbergen, C. J. V. (1979). *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition.

Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. (2008). LabelMe: A Database and Web-Based Tool for Image Annotation. *Int. J. Comput. Vision*, 77(1-3):157–173.

Saad, Y., editor (2011). *Numerical Methods for Large Eigenvalue Problems, 2nd revised edition*. SIAM.

Schapire, R. E. and Freund, Y. (2012). *Boosting: Foundations and Algorithms*. The MIT Press.

Schönemann, P. (1966). A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10.

Sebastiani, F. (2002). Machine Learning in Automated Text Categorization. *ACM Comput. Surv.*, 34(1):1–47.

Shakhnarovich, G., Darrell, T., and Indyk, P. (2006). *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing)*. The MIT Press.

Shakhnarovich, G., Viola, P., and Darrell, T. (2003). Fast Pose Estimation with Parameter-Sensitive Hashing. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 750–, Washington, DC, USA. IEEE Computer Society.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2007). Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 807–814, New York, NY, USA. ACM.

Shen, F., Shen, C., Liu, W., and Tao Shen, H. (2015). Supervised Discrete Hashing. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 37–45.

Shi, J. and Malik, J. (2000). Normalized Cuts and Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905.

Smeulders, A. W. M., Worring, M., Santini, S., Gupta, A., and Jain, R. (2000). Content-Based Image Retrieval at the End of the Early Years. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(12):1349–1380.

Smucker, M. D., Allan, J., and Carterette, B. (2007). A Comparison of Statistical Significance Tests for Information Retrieval Evaluation. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 623–632, New York, NY, USA. ACM.

Song, J., Yang, Y., Yang, Y., Huang, Z., and Shen, H. T. (2013). Inter-media Hashing for Large-scale Retrieval from Heterogeneous Data Sources. In *Proceedings of the*

2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13, pages 785–796, New York, NY, USA. ACM.

Szu, H. and Hartley, R. (1987). Fast simulated annealing. *Physics Letters A*, 122(3-4):157 – 162.

Terasawa, K. and Tanaka, Y. (2007). Spherical LSH for Approximate Nearest Neighbor Search on Unit Hypersphere. In Dehne, F., Sack, J.-R., and Zeh, N., editors, *Algorithms and Data Structures*, volume 4619 of *Lecture Notes in Computer Science*, pages 27–38. Springer Berlin Heidelberg.

Torralba, A., Fergus, R., and Weiss, Y. (2008). Small codes and large image databases for recognition. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*.

Ture, F., Elsayed, T., and Lin, J. (2011). No Free Lunch: Brute Force vs. Locality-sensitive Hashing for Cross-lingual Pairwise Similarity. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 943–952, New York, NY, USA. ACM.

Turpin, A. and Scholer, F. (2006). User Performance Versus Precision Measures for Simple Search Tasks. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 11–18, New York, NY, USA. ACM.

Ulz, M. H. and Moran, S. J. (2013). Optimal kernel shape and bandwidth for atomistic support of continuum stress. *Modelling and Simulation in Materials Science and Engineering*, 21(8):085017.

van der Maaten, L. and Hinton, G. E. (2008). Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Wang, J., Kumar, O., and Chang, S. (2010a). Semi-supervised hashing for scalable image retrieval. In *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, pages 3424–3431.

Wang, J., Kumar, S., and Chang, S. (2010b). Sequential Projection Learning for Hashing with Compact Codes. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 1127–1134.

Wang, J., Kumar, S., and Chang, S.-F. (2012). Semi-Supervised Hashing for Large-Scale Search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(12):2393–2406.

Wang, J., Shen, H. T., Song, J., and Ji, J. (2014). Hashing for Similarity Search: A Survey. *CoRR*, abs/1408.2927.

Wang, Z., Duan, L.-Y., Lin, J., Wang, X., Huang, T., and Gao, W. (2015). Hamming Compatible Quantization for Hashing. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI '15.

Weber, R., Schek, H.-J., and Blott, S. (1998). A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Weiss, Y., Torralba, A., and Fergus, R. (2008). Spectral Hashing. In *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 1753–1760.

Wilcoxon, F. (1945). Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83.

Williams, C. K. I. and Seeger, M. (2001). Using the Nyström Method to Speed Up Kernel Machines. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press.

Wu, B., Yang, Q., Zheng, W.-S., Wang, Y., and Wang, J. (2015). Quantized Correlation Hashing for Fast Cross-modal Search. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI '15, pages 3946–3952. AAAI Press.

Xu, D., Cham, T. J., Yan, S., Duan, L., and Chang, S.-F. (2010). Near Duplicate Identification With Spatially Aligned Pyramid Matching. *IEEE Transactions on Circuits and Systems for Video Technology*, 20:1068–1079.

Xu, H., Wang, J., Li, Z., Zeng, G., Li, S., and Yu, N. (2011). Complementary Hashing for Approximate Nearest Neighbor Search. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 1631–1638, Washington, DC, USA. IEEE Computer Society.

Yang, Y. and Webb, G. I. (2009). Discretization for naive-Bayes Learning: Managing Discretization Bias and Variance. *Mach. Learn.*, 74(1):39–74.

Yuille, A. L. and Rangarajan, A. (2003). The Concave-convex Procedure. *Neural Comput.*, 15(4):915–936.

Zanzotto, F. M., Pennacchiotti, M., and Tsioutsiouliklis, K. (2011). Linguistic Redundancy in Twitter. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 659–669, Stroudsburg, PA, USA. Association for Computational Linguistics.

Zhang, D., Wang, J., Cai, D., and Lu, J. (2010a). Extensions to Self-Taught Hashing: Kernelisation and Supervision. In *Proceedings of the Feature Generation and Selection for Information Retrieval Workshop*, SIGIR '10.

Zhang, D., Wang, J., Cai, D., and Lu, J. (2010b). Self-taught Hashing for Fast Similarity Search. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 18–25, New York, NY, USA. ACM.

Zhang, K., Lan, L., Wang, Z., and Moerchen, F. (2012). Scaling up Kernel SVM on Limited Resources: A Low-rank Linearization Approach. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2012, La Palma, Canary Islands, April 21-23, 2012*, pages 1425–1434.

Zhen, Y. and Yeung, D. (2012). Co-Regularized Hashing for Multimodal Data. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1385–1393.

Zhu, H., Long, M., Wang, J., and Cao, Y. (2016). Deep Hashing Network for Efficient Similarity Retrieval. In *Association for the Advancement of Artificial Intelligence*, AAAI16. AAAI Press.

Zhu, X. and Ghahramani, Z. (2002). Learning from Labeled and Unlabeled Data with Label Propagation. Technical report, Technical Report CMU-CALD-02-107, Carnegie Mellon University.