# 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 $\mathbf{t}_k = [t_{k1}, t_{k2}, \ldots, t_{kT}]$ where $t_{ki} \in \mathbb{R}$ and $t_{k1} < t_{k2} \ldots < t_{kT}$ for each of the $K$ projected dimensions $\left\{\mathbf{y}^k \in \mathbb{R}^N\right\}_{k=1}^K$. 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 $\varepsilon$-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 $\varepsilon$-*NN* groundtruth paradigm and so each pairwise relationship between two

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

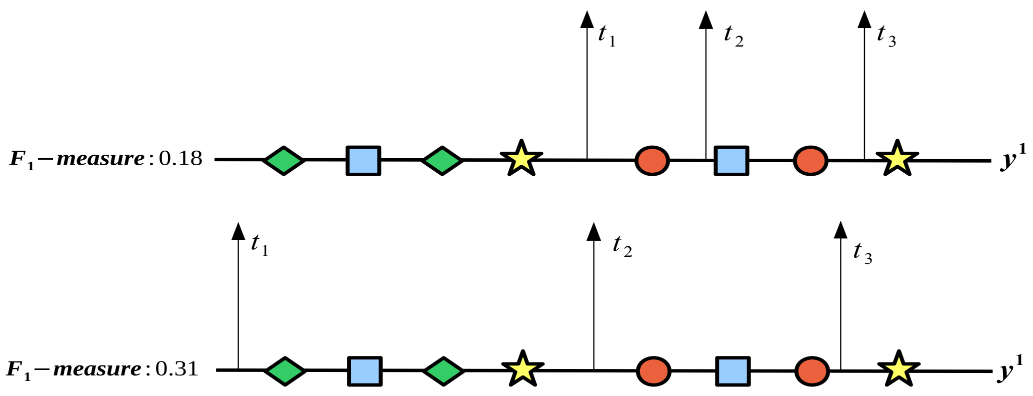


Figure 4.1: Maximisation of the $F_1$-measure can lead to an effective setting of the quantisation thresholds. In both diagrams we seek to position three thresholds along the same projected dimension. In the top diagram the threshold positioning leads to an $F_1$-measure of 0.18. This is a rather low 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 $F_\beta$-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 $\mathbf{t}_k^* = [t_{k1} \ldots t_{kT}]$ 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 $m \in \{1, \dots, M\}$. 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 $\Delta$ 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 $\mathbf{t}_k = [t_1^k \dots t_T^k]$, $\mathbf{t}_k^{'} = [t_1^{k'} \dots t_T^{k'}]$, 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: $\mathbf{t}_k = [t_1^k \dots t_i^k, t_{i+1}^{k'} \dots t_T^{k'}]$ and $\mathbf{t}_k^{'} = [t_1^{k'} \dots t_i^{k'}, t_{i+1}^k \dots t_T^k]$. 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 $\omega H$ 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 $\alpha$. 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 $\alpha$, 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 $\alpha$ 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

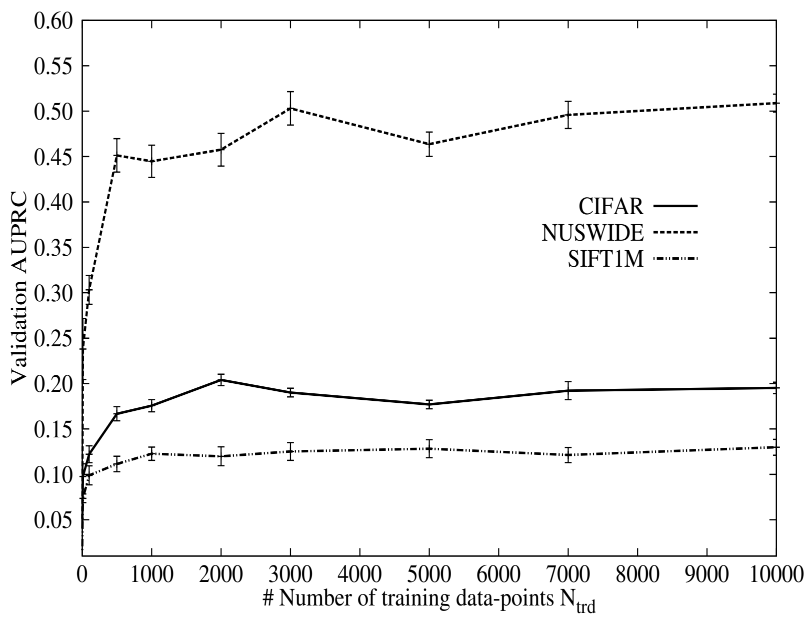


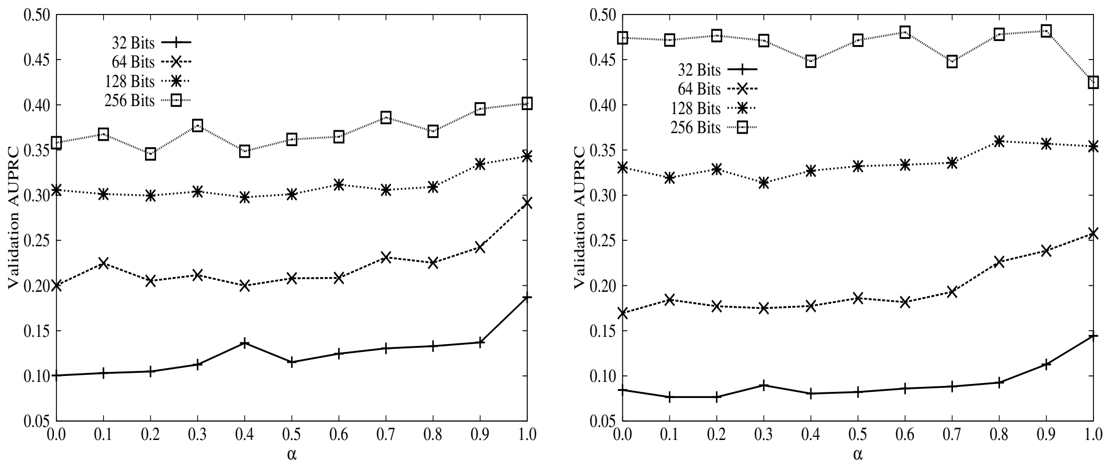
Figure 4.2: The effect of the amount of supervision $N_{trd}$ on the *validation* dataset retrieval effectiveness (AUPRC). The results are shown for LSH projections with $T = 1$ threshold per projected dimension (hashcode length of 32 bits). The bars show the standard error of the mean.

$N_{trd} = 2,000$-$10,000$ 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 $\alpha$ 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 $\alpha$ 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-

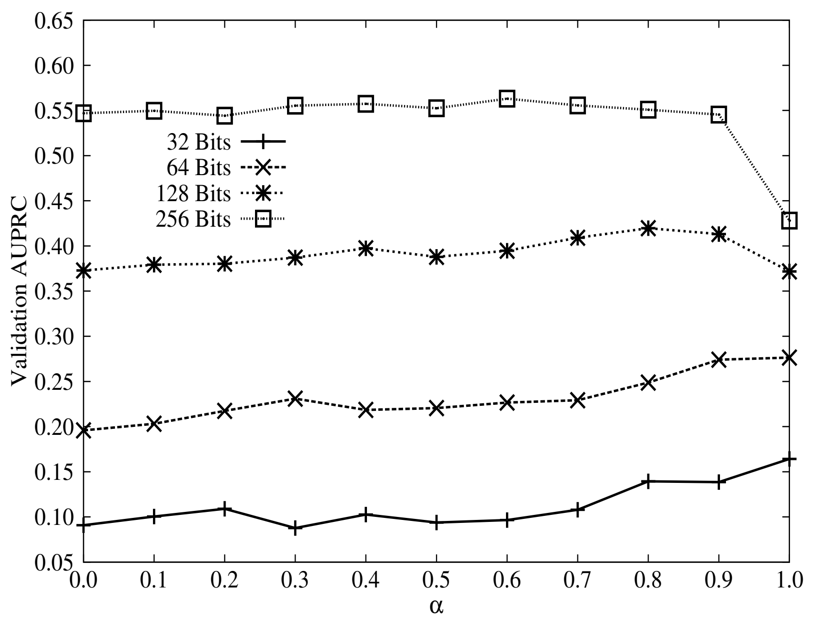
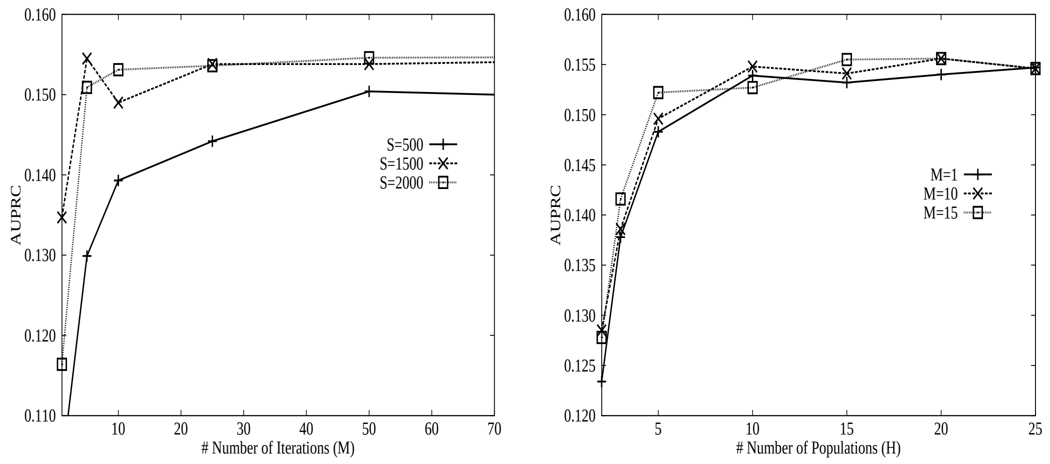


Figure 4.4: Variation in AUPRC with the value of α for the CIFAR-10 dataset and the MHQ codebook (LSH projections). $T = 3$ thresholds are optimised per projected dimension.

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



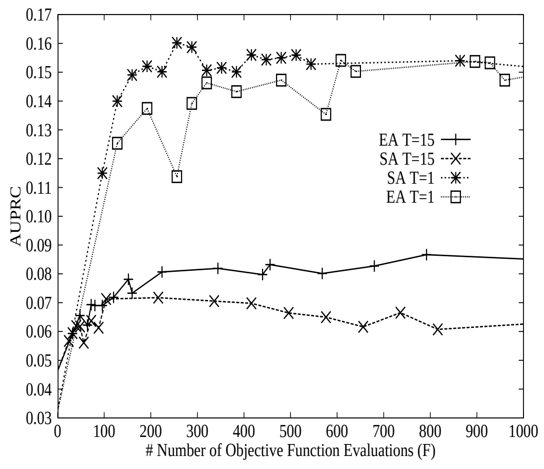
(a) **Simulated Annealing** (b) **Evolutionary Algorithms**

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).

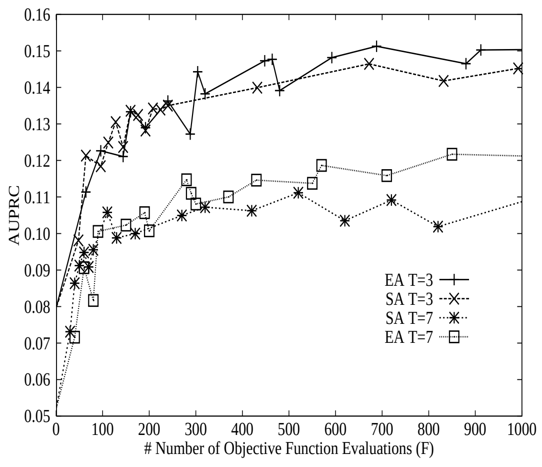
the task of threshold optimisation. Ideally we would like the stochastic search to find a threshold configuration that leads to the greatest AUPRC while taking a minimal number of evaluations of Equation 4.6.

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 $m \in \{1,\dots,M\}$. 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 ($\phi$), crossover ($\theta$) 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** | $\mathbf{T=1}$ | $\mathbf{T=2}$ | $\mathbf{T=3}$ | $\mathbf{T=7}$ | $\mathbf{T=15}$ |
| 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

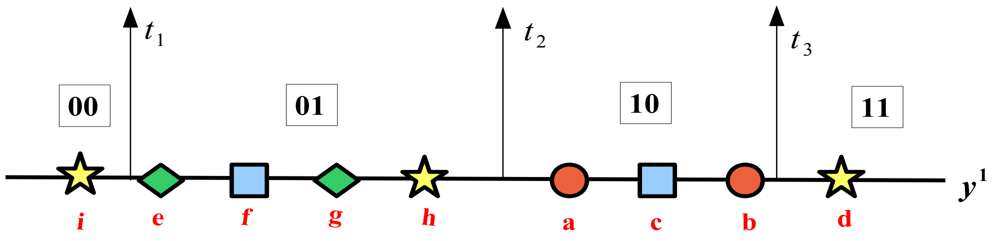


Figure 4.7: Example projected dimension used to illustrate an efficient method of computing the TPs, FPs and FNs required to compute the $F_1$-measure in Equation 4.6.

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.

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

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) |

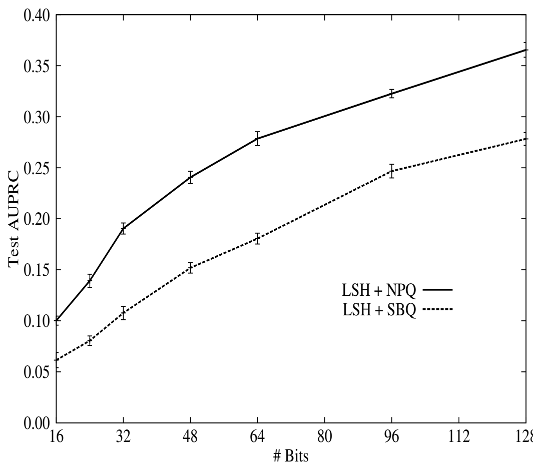
Table 4.8: AUPRC for the single threshold ($T = 1$) optimisation experiment at 32 bits for LSH projections ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over SBQ. The improved splitting strategy result are in brackets.

| | **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) |

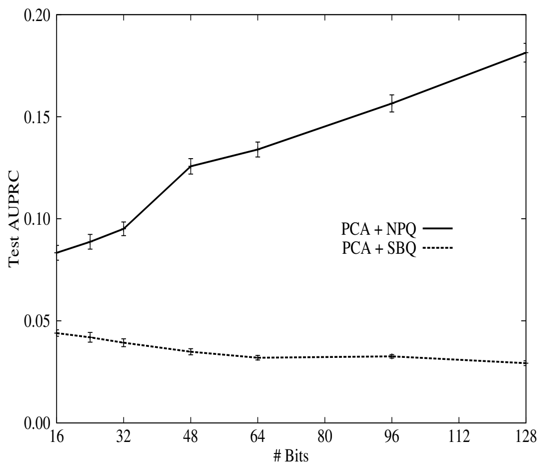
Table 4.9: AUPRC for the single threshold ($T = 1$) optimisation experiment at 32 bits for PCA projections. ▲▲/▼▼ indicates a statistically significant increase/decrease (Wilcoxon, $p < 0.01$) over SBQ. The improved splitting strategy result are in brackets.

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



(a) **LSH projections**



(b) **PCA projections**

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 $\alpha$ 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.

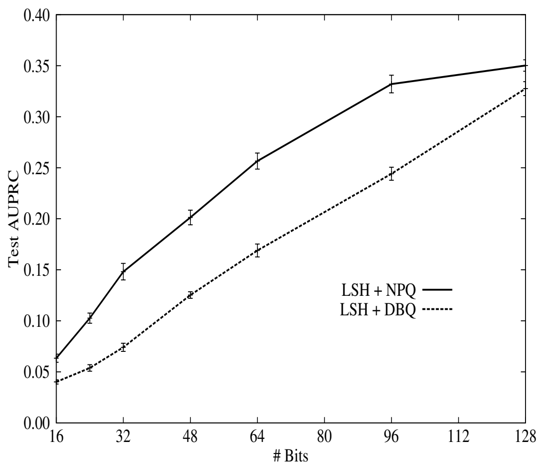
| | **CIFAR-10** | **NUS-WIDE** | **SIFT1M** |
|---|---|---|---|
| LSH + NPQ | **0.1535▲▲** | **0.3459** | 0.0933 |
| LSH + DBQ | 0.0786 | 0.1460 | 0.0764 |
| LSH + SBQ | 0.1069 | 0.3395 | **0.0974** |
| LSH + EQL | 0.0342 | 0.0228 | 0.0208 |
| LSH + RND | 0.0319 | 0.0188 | 0.0062 |

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.

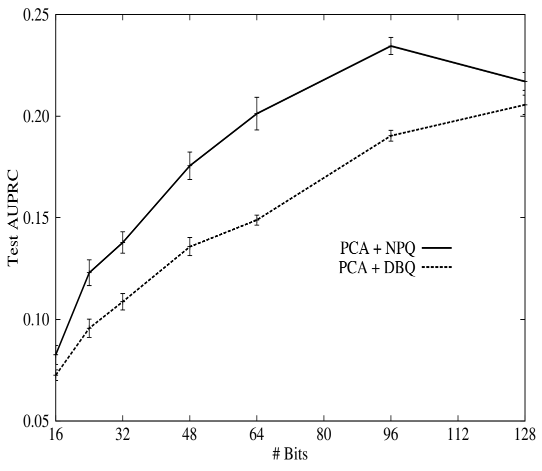
Examining the results in Table 4.11 we observe that the multi-threshold optimisation model (LSH+NPQ) attains a statistically significant increase (Wilcoxon signed rank, $p < 0.01$) in retrieval effectiveness with respect to the DBQ, EQL and RND baseline quantisation algorithms across all three still 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 **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 |

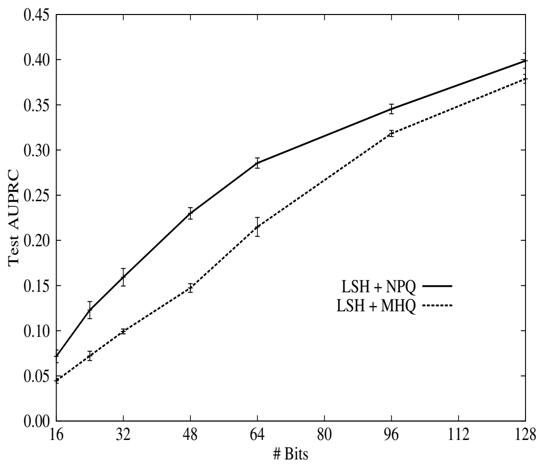
Table 4.16: Parametrisation of the quantisation methods studied in experiment IV. NBC stands for natural binary code.

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.

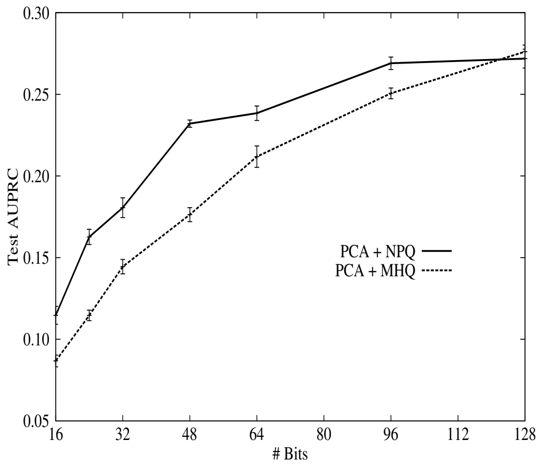
#### 4.3.3.8 Experiment IV: Generalisation to other Projection Functions

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

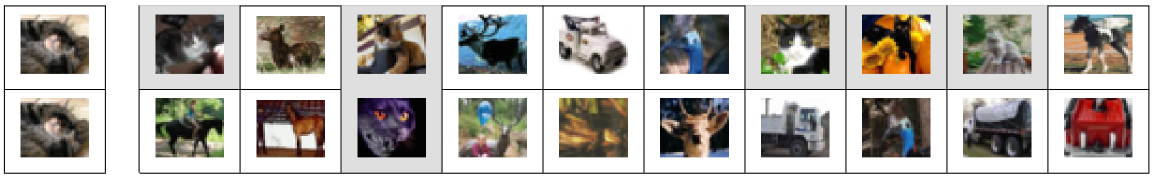


Table 4.20: **Left-most column:** Cat query image. **Top row:** ITQ+NPQ top 10 retrieved images, precision: 0.5. **Bottom row:** ITQ+SBQ top 10 retrieved images, precision: 0.1. Shaded cells indicate true positives.

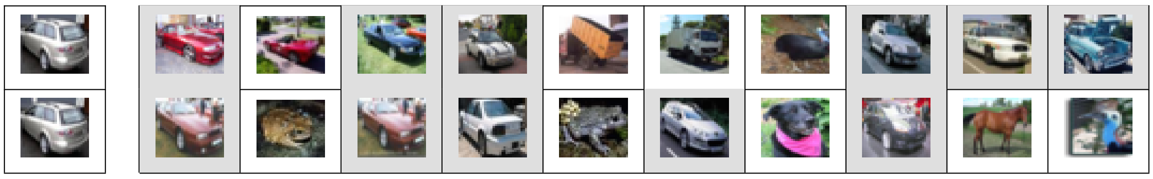


Table 4.21: **Left-most column:** Car query image. **Top row:** ITQ+NPQ top 10 retrieved images, precision: 0.7. **Bottom row:** ITQ+SBQ top 10 retrieved images, precision: 0.5.

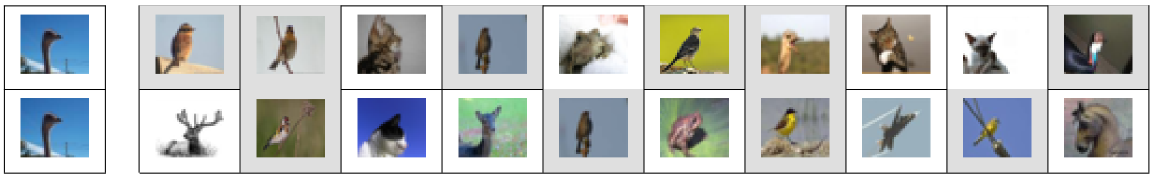


Table 4.22: **Left-most column:** Bird query image. **Top row:** ITQ+NPQ top 10 retrieved images, precision: 0.6. **Bottom row:** ITQ+SBQ top 10 retrieved images, precision: 0.4.

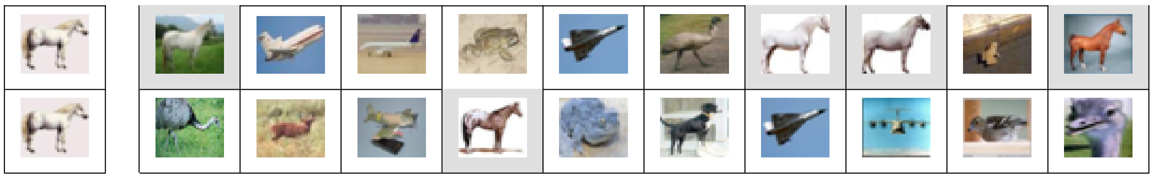


Table 4.23: **Left-most column:** Horse query image. **Top row:** ITQ+NPQ top 10 retrieved images, precision: 0.4. **Bottom row:** ITQ+SBQ top 10 retrieved images, precision: 0.1.

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-dependent (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.