Algorithms for large-scale multi-codebook quantization

by

Julieta Martinez-Covarrubias

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF SCIENCE
(Computer Science)

The University of British Columbia
(Vancouver)

December 2018

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Algorithms for large-scale multi-codebook quantization

submitted by **Julieta Martinez-Covarrubias** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** in **Computer Science**.

Examining Committee:

Prof. James J. Little, Computer Science, University of British Columbia
Co-supervisor

Prof. Holger H. Hoos, Computer Science, University of British Columbia
Co-supervisor

Prof. Mark Schmidt, Computer Science, University of British Columbia
Supervisory Committee Member

Prof. Michiel van de Panne, Computer Science, University of British Columbia
University Examiner

Prof. Vincent Wong, Electrical Engineering, University of British Columbia
University Examiner

Prof. David J. Fleet, Computer Science, University of Toronto
External Examiner

Abstract

Combinatorial vector compression is the task of expressing a set of vectors as accurately as possible in terms of discrete entries in multiple bases. The problem is of interest in the context of large-scale similarity search, as it provides a memory-efficient, yet ready-to-use compact representation of high-dimensional data on which vector similarities such as Euclidean distances and dot products can be efficiently approximated.

Combinatorial compression poses a series of challenging optimization problems that are often a barrier to its deployment on very large scale systems (e.g., of over a billion entries). In this thesis we explore algorithms and optimization techniques that make combinatorial compression more accurate and efficient in practice, and thus provide a practical alternative to current methods for large-scale similarity search.

Lay Summary

Imagine someone had a long list of every post office in the country and wanted to figure out which one to use. If they know the location of their house they could calculate the distance to each post office in the list, keep track of the minimum distance, and easily find their nearest post office. This problem is known as "nearest neighbour search".

Many problems in computer science involve solving this problem many times per second, in high dimensions, and on very large databases. This thesis introduces a series of algorithmic improvements for systems that perform this task approximately on very large datasets, trading off accuracy, memory usage, and speed.

Preface

The main work presented in this thesis is based on three publications.

- A version of Chapter 3 has been published in the Proceedings of the European Conference on Computer Vision (ECCV) (Martinez et al., 2016a):

  > J. Martinez, J. Clement, H. H. Hoos, and J. J. Little. Revisiting additive quantization. In *ECCV 2016*.

  and in (Martinez et al., 2016b)

  > J. Martinez, H. H. Hoos, and J. J. Little. Solving multi-codebook quantization in the GPU. In *4th Workshop on Web-scale Vision and Social Media (VSM), ECCV workshops, 2016*.

  The chapter and the CUDA code were both written by Martinez. Clement helped run experiments and wrote Julia code for the software release related to the above publications.

- A version of Chapter 4 has been published in the Proceedings of the European Conference on Computer Vision (ECCV) (Martinez et al., 2018):

  > J. Martinez, S. Zakhmi, H. H. Hoos, and J. J. Little, LSQ++: Lower running time and higher recall in multi-codebook quantization. In *ECCV 2018*.

  The chapter was entirely written by Martinez. Zakhmi helped run experiments and write Julia code for the new algorithms described in Chapter 4.

- Chapter 6 was written by Martinez, and the work presented there was developed by Martinez and Melody Wong during her internship at the UBC vision lab in the summer of 2017.
- Chapters 5-7 contain new material. Hoos, Little and Mark Schmidt have edited and revised all the chapters in this thesis.

Table of Contents

Glossary

ANN	Approximate nearest neighbour
ANNS	Approximate nearest neighbour search
API	Application programming interface
AQ	Additive quantization
CKM	Cartesian *k*-means
CG	Conjugate gradient, a family of methods for solving large-scale systems of linear equations.
CNN	Convolutional neural network
CQ	Composite quantization
COMPQ	Competitive quantization
ERVQ	Enhanced residual vector quantization
EM	Expectation-maximization, an optimization approach for problems with multiple latent variables
FAISS	Facebook AI Similarity Search, a library for large-scale approximate nearest neighbour search
FLANN	Fast Library for Approximate Nearest Neighbours, a popular library for fast approximate nearest neighbour search

FALCONN Fast Lookups of Cosine and Other Nearest Neighbors, a library for approximate nearest neighbour search

GPU Graphics processor unit

HDD Hard-drive disk

ICM Iterated conditional modes, an approximate method for maximum-a-posteriori (MAP) estimation in Markov random fields (MRF)

ILS Iterated local search

ILSRVC ImageNet Large-Scale Visual Recognition Challenge, a benchmark to classify images into 1 000 categories.

JKM Joint k-means quantization

LBP Loopy belief propagation

L-BFGS Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm, a method for constrained optimization

LSH Locallity-sentitive hashing

LSQ Local search quantization, a multi-codebook quantization method introduced in this thesis

MAP Maximum-a-posteriori, the most likely state in a Markov random field (MRF)

MRF Markov random field

MCQ Multi-codebook quantization

MIPS Maximum inner product search

NNS Nearest neighbour search

OCKM Optimized Cartesian k-means

OPQ Optimized product quantization

OTQ Optimized tree quantization

PCA Principal component analysis

PQ Product quantization

RAM Random access memory

RL Reinforcement learning

RVQ Residual vector quantization

SAT Boolean satisfiability problem

SCQ Sparse composite quantization

SA Simulated annealing

SGD Stochatic gradient descent

SH Spectral hashing

SIMD Single instruction multiple data

SLS Stochastic local search

SOBE Self-organizing binary encoding

SMAC Sequential model-based optimization for algorithm configuration, a method for automatic hyperparameter optimization.

SPGL1 Spectral projected gradient, a method for large-scale, ℓ_1-regularized, least-squares optimization

SSD Solid-state drive

SQ Stacked quantizers

SR Stochastic eelaxation

SR-D An extension of simulated annealing applicable to multi-codebook quantization, introduced in this work

SR-C An extension of simulated annealing applicable to multi-codebook quantization, introduced in this work

SIFT Scale invariant feature transform, a popular local image descriptor

TC Transform coding

TSP Travelling salesperson problem

VQ Vector quantization

Acknowledgements

I would like to thank my supervisors, Prof. James J. Little and Prof. Holger H. Hoos, for their advice and support during the past five years. The ideas presented in this thesis sparked from conversations with them, and the development work would not have been possible without their support. I also want to thank Prof. Mark Schmidt for his input on this work as a member of my thesis committee, and for constantly pushing me to learn new topics and improve my presentation skills in the machine learning reading group at UBC. I am also thankful to Prof. David Fleet, Prof. Michiel van de Panne, and Prof. Vincent Wong for their thoughtful comments on this work.

I am also grateful to Javier Romero, Prof. Michael J. Black, and the rest of the team at the Max Planck Institute for Intelligent Systems, whom I was fortunate to work with during the fall of 2016.

I am also indebted to my interns, Joris Clement, Shobhit Zakhmi, and Melody Wong, for trusting me as a mentor and allowing me to learn so much from them.

Thanks also to Anahi Molar Cruz, Bita Nejat, Erica Peredo, Jaimie Veale, Jennifer Gordon, Sarina McFarland, Sampoorna Biswas, Shay Loo, Shoshana Izsak, and Silvia Picazo Barragán. Munich, San Diego, Aotearoa, Mexico City, Eindhoven, Vancouver and Toronto would not have been the same without you. I love all of you, and I cannot believe how fortunate I am to have you in my life.

Finally, I acknowledge that this thesis was completed while I lived and worked at UBC, which is located in the traditional, ancestral, and unceded territory of the Musqueam people.

Chapter 1

Introduction

A notable characteristic of the computational landscape at the turn of the 21st century is an unprecedented growth in the use and collection of data. Scientific fields such as genetics, bioinformatics, chemistry, astronomy, particle physics and climate science have all benefited from large-scale data collection mechanisms. At the same time, large-scale data collection has inspired new research directions in those areas.

In everyday life, a data-collection explosion has appeared in a more familiar form. Consumer-grade cameras (which are often attached to smartphones), coupled with the growth in popularity of social media platforms and increased interconnectivity, have made it very easy for the average consumer in developed nations to record and store visual narratives at an unprecedented rate. From the perspective of a smartphone owner, taking a picture or recording a short video of a person, event or object that they care about, and sharing this content with their social network, is a now a common, easy, and low-overhead task.

At the same time, deep learning techniques have, over the past decade, dramatically improved the state of the art on a variety of computational perceptual tasks such as speech and object recognition. Such techniques typically consist of a multilayered function approximator that produces, as an end result, a high-dimensional real-valued vector that is amenable to tasks such as multi-label classification. Systems that perform tasks such as object classification can now be used to extract, if at a somewhat basic level, semantic information from large collections of images

and video.

A problem that occurs in practice is creating an effective tool to search for objects or events of interest in a large collection of visual data. Imagine, for example, building a search engine that can be queried with other images, and returns an ordered list of semantically similar pictures to the user. While images can be preprocessed by deep architectures, one still needs to build an indexing system that is able to compute similarities in a high-dimensional space and scales graciously in terms of memory (because data keeps growing), speed (because users expect fast answers), and accuracy (because users expect correct answers).

The goal of this thesis is to explore algorithms and optimization techniques that are useful to build systems for vector similarity search and scale well with data growth. We focus on applications in computer vision, but the fundamental problem addressed is large-scale similarity search, of which maximum inner product search (MIPS) and approximate nearest neighbour search (ANNS) are special cases. These two problems have multiple applications across computer science.

Chapter 2 includes more detail on the history and applications of the problems in general, and here we include a small introduction in the context of computer vision.

1.1 Multi-codebook quantization (MCQ) applications in computer vision

Many computer vision applications involve computing the similarity of many high-dimensional, real-valued image representations, in a process known as feature matching. For example, in structure from motion (Snavely et al., 2006), it is common to estimate the relative viewpoint of each image in large collections of photographs by computing the pairwise similarity of several million scale invariant feature transform (SIFT) (Lowe, 2004) descriptors. It is also now common for retrieval and classification datasets to comprise millions of images (Deng et al., 2009; Torralba et al., 2008), and for deep learning pipelines to directly learn representations tailored for retrieval on millions of images (Gordo et al., 2017). In a commercial application, the popular reverse image search engine TinEye (https://www.TinEye.com/) currently searches over 27.6 billion images indexed from all

over the Internet. When such large databases of images are used, matching poses significant computational bottlenecks.

In practice, the large-scale matching problem often translates into large-scale ANN search, a problem that has traditionally been addressed with hashing (Gong and Lazebnik, 2011; Weiss et al., 2009). However, a series of methods based on vector quantization have recently demonstrated superior performance and scalability, sparking interest from the machine learning, computer vision and multimedia retrieval communities (Babenko and Lempitsky, 2014a; Ge et al., 2014; Jégou et al., 2011; Norouzi and Fleet, 2013; Zhang et al., 2014). These methods are all based on multi-codebook quantization (MCQ), a generalization of k-means clustering with cluster centres arising from the sum of entries in multiple codebooks. Other applications of MCQ include large-scale MIPS (Guo et al., 2016; Shrivastava and Li, 2014), and the compression of deep neural networks for mobile devices (Wu et al., 2016).

1.2 Optimization challenges in MCQ

Like k-means clustering, MCQ is posed as the search for a set of codes and codebooks that best approximate a given dataset. Importantly, the error of this approximation provides a lower bound for Euclidean distance and dot-product approximations in ANNS and MIPS applications – and, by extension, in convolution approximations, as used by Wu et al. (2016) to compress convolutional neural networks (CNNs). Therefore, finding optimization methods that achieve low-error solutions is crucial for improving the performance of MCQ applications.

Early approaches to MCQ were designed enforcing codebook orthogonality (Ge et al., 2014; Jégou et al., 2011; Norouzi and Fleet, 2013), which greatly simplifies the problem at the expense of accuracy. In contrast, more recent methods use non-orthogonal, often full-dimensional codebooks (Babenko and Lempitsky, 2014a, 2015; Ozan et al., 2016a; Zhang et al., 2014), and tend to be more computationally expensive than their classical counterparts, which in practice limits their applicability on very large-scale datasets. Thus, it is also important to find efficient optimization methods that scale well with dataset size.

1.3 Contributions to MCQ outlined in this thesis

In this thesis, we explore algorithms for MCQ and their applications in ANNS. We make several contributions to this problem:

- Fixing the codebooks in an MCQ problem produces a distribution of small, yet hard combinatorial optimization problems to find the optimal corresponding codes. We have noticed that there is a large field of literature with corresponding well-established techniques for designing algorithms that target distributions of NP-hard problems, such as the Boolean satisfiability problem (SAT) instances that arise in particular industrial applications. Inspired by this area of research, in Chapter 3 we describe an algorithm based on iterated local search (ILS) for finding these codes in MCQ, and detail a graphics processor unit (GPU) implementation that greatly accelerates this part of MCQ optimization.

- If one fixes the codes in an MCQ problem, finding the optimal codebook becomes a large-scale least squares problem. Previous work has proposed using large-scale conjugate-gradient solvers for this step (Babenko and Lempitsky, 2014a), or used decomposition-based direct methods that, nonetheless, require expensive matrix multiplications (Zhang et al., 2014). In Chapter 4 we exploit the structure of the code matrix and propose a direct least-squares method that can be efficiently computed via Cholesky decomposition on a small, symmetric, positive definite matrix. Conveniently, this matrix can be computed using the codes as indices, i.e., without an explicit matrix multiplication routine. This results in a method for codebook update that is orders of magnitude faster than previous work.

- We note that MCQ is much like k-means clustering, except that it is combinatorial. However, if one finds methods that improve k-means, one might be able to adapt them to improve MCQ as well. In Chapter 4, we introduce *stochastic relaxations* (SR), two methods inspired in simulated annealing used in the 1990s to improve k-means clustering, and adapt them to MCQ, which results in better distance approximations at a negligible increase in computation.

- Our novel encoding algorithm and our SR methods have a number of hyperparameters that influence the performance of our algorithms in practice. We automatically search for good hyperparameters using an automatic algorithm configuration procedure in Chapter 5.

We also attempted to use deep reinforcement learning to learn the parameters of a (deep) encoding function for MCQ. However, our attempts did not result in an algorithm with better computation-to-recall trade-offs than those we achieved with ILS. Therefore, we consider these experiments to be a negative result. We report our setup and experimental results in Chapter 6.

1.4 The Rayuela.jl library

Unfortunately, most of the work reviewed in this thesis, which includes baselines for our work, has been published without giving access to public code that can be used to reproduce its results. This, in practice, puts the burden of proof on other researchers, because in order to show that they have made progress on a problem, they also have to reproduce previous baselines. This is a daunting task that consumed a large part of the time we spent working on this problem.

Together with this thesis we are releasing Rayuela.jl, a software package for large-scale similarity search written in Julia (Bezanson et al., 2014). Rayuela.jl is available under an MIT licence, and can be downloaded at https://github.com/una-dinosauria/Rayuela.jl. Rayuela implements multiple baselines and algorithms for MCQ, and is crucial in showing that our contributions result in Pareto improvements with respect to previous work. We give further details about the development of Rayuela.jl and future work in Chapter 7.

Rayuela.jl is our attempt to make this research reproducible and accessible, and to make it easier for others, especially newcomers, to contribute to this field. We also release it with the hope of alleviating the irreproducibility cycle that currently plagues computer vision research.

Chapter 2

Background

Used to be so wilfully obtuse
Or is the word abstruse?
Semantics like a noose
Get out your dictionaries!
—Andrew Bird

In this thesis we focus on techniques for approximate nearest neighbour search (ANNS) that are based on vector quantization (VQ). First, we provide some context on approaches for (exact) nearest neighbour search (NNS), and then discuss methods for ANNS. Finally, we focus on VQ techniques for ANNS.

2.1 Nearest neighbour search (NNS)

Given a dataset X comprised of n entries $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}, \mathbf{x}_i \in \mathbb{R}^d$, a query $\mathbf{q} \in \mathbb{R}^d$, and a distance function $dist : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ the nearest neighbour search problem is the task of finding an element in $NN(\mathbf{q}, X) \in X$ such that

$$NN(\mathbf{q}, X) \in \operatorname*{argmin}_{\mathbf{x} \in X} dist(\mathbf{q}, \mathbf{x}), \tag{2.1}$$

usually, $dist(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|_2$ (i.e., Euclidean distance). For notational convenience, we often represent $X \in \mathbb{R}^{d \times n}$ as a matrix, where we simply stack all the d-dimensional database vectors $\mathbf{x}_i$ horizontally.

Running time in nearest neighbour search (NNS)

There are two main phases in ANN algorithms where running time is important. Offline time, also called training time, includes the time that is spent preprocessing X. Online time, also called query time, includes the time that, given a query, it takes to obtain k of its neighbours in a dataset.

While training time is less critical than query time, everything else being equal it is clearly desirable to have higher approximation accuracy and shorter training times. Because in practice we often deal with very large datasets, a difference of minutes to hours in training time on a small dataset can mean a difference from days to months on larger datasets.

Query time is more critical, because it is often the interface that the user observes. For example, if our ANNS system forms the backend of an image search engine or a recommendation system, a difference of milliseconds in response time could result in a vastly different experience to the end user.

2.1.1 Exact NNS algorithms

Exhaustive search

A trivial solution consists of exhaustively computing $dist(\mathbf{q}, \mathbf{x}_i)$ for all vectors $\mathbf{x}_i$ in X, and keeping track of the entry that produces the minimum distance. This algorithm, although exact, is also potentially very inefficient for large n or large d. The research challenge is thus to find ways to preprocess X, such that we can find neighbours efficiently.

Very low dimensions ($d = \{1, 2\}$)

For the special case when $d = 1$, sorting X and performing binary search guarantees finding the nearest neighbour in $O(\log n)$ query time. This technique is typically used as a subroutine of sorting algorithms and data structures such as heaps and ordered queues. For the special case when $d = 2$, it is possible to perform exact search in $O(d \cdot \log n)$ time using Voronoi diagrams (Aurenhammer, 1991).

Higher dimensions ($d \geq 3$)

Unfortunately, no exact NNS algorithms with $O(d \cdot \log n)$ query time complexity are known for dimensionality $d \geq 3$.

Kd-trees (Bentley, 1975) recursively partition the space across dimensions and return a correct solution every time, but only have an $O(d \cdot \log n)$ expected, not guaranteed, query time. Similarly, Ball trees (Omohundro, 1989) partition the space in a series of potentially overlapping hyperspheres, and provide an expected $O(d \cdot \log n)$ query time. Finally, Norouzi et al. (2014) introduce a method for exact search with expected sub-linear search time, but their results is only applicable to Hamming space. These methods, however, do not work well for high dimensions (Kibriya and Frank, 2007; Muja and Lowe, 2009). In this case, it is common to resort to approximate algorithms.

2.2 Approximate nearest neighbour search (ANNS)

We now review ANNS algorithms, dividing them into memory-oblivious (i.e., ones that do not reduce storage), and memory-conscious. The latter scale better with data growth, and are the main focus of this thesis.

2.3 Memory-oblivious ANNS techniques

A modern commodity computer handles memory in a hierarchy, trading off capacity for speed of access. In a desktop server, a hard-drive disk (HDD) or a solid-state drive (SSD) is often the largest and slowest unit of memory, currently in the order of a Terabyte. Next is the random access memory (RAM), several orders of magnitude faster than main storage, but also more scarce, currently in the order of 32-64 gigabytes per machine. Closer to the processor are several levels of cache memories, typically in the order of a few megabytes to hundreds of kilobytes, followed by the fastest – and smallest – registers, with a few bytes of capacity each.

Typically, operating systems allow user programs to store their state and variables in RAM, as disk accesses are considered too slow for user interaction. Similarly, most research in methods for ANNS consider datasets small enough in magnitude to be fully stored in RAM, and often build data structures that require additional

memory on top of the dataset itself. These memory requirements clearly impose practical limitations and the size of datasets that a user can handle. For this reason, we refer to these methods as "memory-oblivious".

Most research in ANNS falls under this category. In the computer vision community, the most popular methods are perhaps randomized kd-trees (Silpa-Anan and Hartley, 2008), and hierarchical k-means clustering (Nister and Stewenius, 2006), both implemented in the very successful Fast Library for Approximate Nearest Neighbours (FLANN) (Muja and Lowe, 2009). More recent data structures and efficient implementations include the Hierarchical Navigable Small World Graphs (Malkov and Yashunin, 2016), or the Fast Lookups of Cosine and Other Nearest Neighbors (FALCONN) library based on an improved version of E2LSH (Andoni et al., 2015). FALCONN, for example, provides theoretical guarantees as well, but as Matthijs Douze points out[1]:

> [Locality sensitive hashing] and its derivatives suffer from two drawbacks: [(1)] They require a lot of hash functions ([number of] partitions) to achieve acceptable results, leading to a lot of extra memory. Memory is not cheap[, and (2) t]he hash function are [sic] not adapted to the input data. This is good for proofs but leads to suboptimal choice results [sic] in practice.

Indeed, for very large datasets, memory is usually the most constrained resource. Conversely, however, theory-driven methods such as Locallity-sentitive hashing (LSH) often provide theoretical bounds on the worst performance of any query.

Quantization-based methods, in general, do not provide such guarantees and in practice assume that the queries come from the same distribution as the training set. This reflects the workflow of applications such as image search or 3d reconstruction, where this assumption is true. Moreover, vector quantization methods are memory-conscious (compressing the data is at their core) and have great performance in practice. We now review work in this area.

[1] https://github.com/facebookresearch/faiss/wiki/Faiss-indexes#relationship-with-lsh

2.4 Multi-codebook quantization (MCQ) for approximate nearest neighbour search

Historically, the first methods for ANNS using MCQ considered only orthogonal codebooks (i.e., methods that encode different subspaces of the data independently). These methods were introduced mainly before 2012, when Scale invariant feature transform (SIFT) descriptors (Lowe, 2004) were the main features used in computer vision tasks. Given a database of images, the object matching pipeline proposed by Lowe consists of (i) offline computing interest points and SIFT descriptors of each image, and (ii) matching features from an unseen image to those in the database – hence the name of the task: *feature matching*.

To match SIFT features, Lowe suggests for each SIFT descriptor in the image, finding the two nearest neighbours in the database, and keeping only the matches whose ratio of the distances to the first and the second neighbours is below 0.8, to ensure discriminability. This came to be known as Lowe's distance criterion.

Given the success of SIFT, and since a high-resolution image can easily produce tens of thousands of descriptors, the computer vision community became interested in finding fast and scalable methods for nearest neighbour search in high dimensions. In this context, orthogonal VQ methods are particularly attractive, not only because they are relatively easy to optimize, but also because they have the advantage that one can compute approximate Euclidean distances using a number of table lookups linear in the number of codebooks.

Orthogonal methods, however, are bound to have fewer parameters than methods with full-dimensional (i.e., non-orthogonal) codebooks. Non-orthogonal methods are conversely harder to optimize, have quadratic query times (a detail that can *and must* be included for a fair comparison), and have overall better recall results when properly optimized. With non-orthogonal methods one can also compute approximate dot products with a number of table lookups linear on the number of codebooks, so they are generally better suited for ANNS under dot product similarity.

2.4.1 Orthogonal methods

Product quantization (PQ)

The first uses of vector compression for ANNS are due to both Jégou et al. (2009) and Sandhawalia and Jégou (2010). These two papers noted that the approximation quality (i.e., the quantization error) of a quantizer bounds the distance approximation of a query, and suggested using lookup tables to speed-up query response time. Jégou et al. (2009) called their method product quantization (PQ). The empirical evaluation compared PQ against spectral hashing (SH) (Weiss et al., 2009), and the Hamming embedding method of Jégou et al. (2008), obtaining considerably higher recall given the same memory usage.

Formally, the goal in PQ is to determine

$$\min_{C,B} \|X - CB\|_F^2, \tag{2.2}$$

where the set to quantize is $X \in \mathbb{R}^{d \times n}$, having n data points with d dimensions each. The latent variables are both the codebooks $C = [C_1, C_2, \ldots, C_m]$, $C_i \in \mathbb{R}^{d \times n}$, which are constrained to be mutually orthogonal (i.e., C is block-diagonal):

$$C = [C_1, C_2, \ldots, C_m] = \begin{bmatrix} \hat{C}_1 & 0 & \ldots & 0 \\ 0 & \hat{C}_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \ldots & \hat{C}_m \end{bmatrix}, \tag{2.3}$$

where, without loss of generality, $\hat{C} \in \mathbb{R}^{\lfloor d/m \rfloor \times h}$ (if d is not divisible by m, the convention is to assign the extra dimensions to the first few codebooks), and the codes $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n] \in \{0,1\}^{m \cdot h \times n}$, where $\mathbf{b}_i = [\mathbf{b}_{i1}^\top, \mathbf{b}_{i2}^\top, \ldots, \mathbf{b}_{im}^\top]^\top \in \{0,1\}^{m \cdot h}$, and each subcode $\mathbf{b}_{ij} \in \{0,1\}^h$ is limited to having only one non-zero entry: $\|\mathbf{b}_{ij}\|_0 = 1$ and $\|\mathbf{b}_{ij}\|_1 = 1$. Figure 2.1 shows these constraints in a more graphical and intuitive way.

Jégou et al. (2009) also introduced the idea of using an inverted file, similar to that proposed by Sivic and Zisserman (2003), to shortlist candidate vectors and further reduce query time, which allowed the system to scale up to one billion

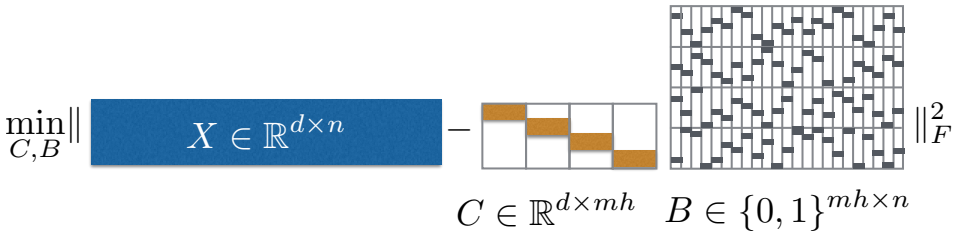

Figure 2.1: Illustration of the PQ optimization problem.

vectors.

On the other hand, Sandhawalia and Jégou did not use product codes, but quantized each dimension independently, and compared only against Spectral hashing (SH). In parallel work, Brandt (2010) proposed a similar method – that quantizes dimensions independently – but also has data-driven bit allocation and non-uniform minimum distortion quantization. Brandt simply calls the method transform coding (TC).

A followup version of the tech report by Jégou et al. (2009) was later published in a computer vision journal (Jégou et al., 2011), and further included a comparison against FLANN (Muja and Lowe, 2009), a library for ANNS that is extremely popular in the computer vision community, obtaining slightly better recall at comparable query times. Since ANNS requires storing the entire database in RAM at runtime, these results demonstrated that PQ with an inverted file would a better option for large-scale ANNS.

Optimized product quantization (OPQ)

Jégou et al. noticed that using either a random permutation of the dimensions, or using a random rotation (i.e., equalizing variance among subspaces) to preprocess the data resulted in better recall rates. Followup work (Jégou et al., 2012) then suggested PCA-rotating the data for dimensionality reduction, and then applying a random orthogonal transformation to equalize variance among subspaces and further improve accuracy. With these observations in mind, Ge et al. (2014) introduced optimized product quantization (OPQ), a method that learns a rotation that both maximizes independence and distributes the variance among the subspaces

of PQ. Norouzi and Fleet (2013) independently discovered the same method, and published it at the same conference, under the name of Cartesian k-means (CKM).

2.4.2 Semi-orthogonal methods

Some vector quantization methods in the ANNS literature use neither fully-independent, nor fully-dimensional codebooks, and thus exist as a middle ground between both ends of the spectrum.

Composite quantization (CQ)

Zhang et al. (2014) propose composite quantization (CQ), a method that optimizes the codebooks such that they are fully-dimensional, but the dot product of any vectors in two different codebooks is close to a constant – the so-called "near-orthogonality" constraint. This results in a non-linear constraint optimization problem that in CQ is solved with the limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm (Nocedal, 1980). Followup work has extended CQ to use sparse codebooks, in a method coined sparse composite quantization (SCQ) (Zhang et al., 2015), and mixed CQ with collaborative filtering to improve cross-modal search (Zhang and Wang, 2016).

Notably, CQ manages to keep query time linear in the number of codebooks, without incurring additional memory cost. This is an impressive feat that no other semi or non-orthogonal method can achieve without using a codebook from the memory budget to store norm approximation. This detail, however, is often omitted in the literature.

Zhang et al. released code to reproduce their results more than 3 years after publication (https://github.com/hellozting/CompositeQuantization/), at the same time that an extended version with journal formatting appeared on arXiv (Wang and Zhang, 2017). Inspecting this code, we have learned that the paper reported results using a non-standard protocol, which follow-up work on non-orthogonal quantization is unaware of. We give more details about this finding in the experimental evaluation of Chapters 3 and 4.

Optimized tree quantization (OTQ)

Babenko and Lempitsky (2015) introduce optimized tree quantization (OTQ), a method that limits the codebook non-orthogonality to either one or two interactions per codebook, and constrains the topology of these interactions to form a tree. While in fully non-orthogonal methods encoding is usually NP-hard, using a tree structure means that one can find optimal encodings in polynomial time using the min-sum (resp. max-product) algorithm. Updating the codebooks, however, results in a binary integer linear program, which in OTQ is solved using the commercial Gurobi solver. A major downside of OTQ is that it increases query time by a factor of two, compared to PQ and OPQ; thus, the results reported in that work are not strictly comparable to these baselines.

Optimized Cartesian k-means (OCKM)

Wang et al. (2014) proposed optimized Cartesian k-means (OCKM) a semi-orthogonal method whose codebooks are non-orthgonal with only one other codebook. Topologically, this results in a forest, rather than a chain or a tree. Like OTQ, this results in increased query time, which was not accounted for during the experimental evaluation comparing against PQ and OPQ.

2.4.3 Non-orthogonal methods

Non-orthogonal MCQ methods use full-dimensional codebooks, thus have more free parameters, and pose more challenging optimization problems. They also tend to deliver lower quantization error, and thus better distance approximations and higher recall. Formally, the goal of non-orthogonal MCQ is to determine

$$\min_{C,B} \|X - CB\|_F^2, \tag{2.4}$$

and the variables are defined just as in PQ, except that the codebooks $C = [C_1, C_2, \ldots, C_m]$, $C_i \in \mathbb{R}^{d \times h}$, are *not* constrained to be mutually orthogonal. Figure 2.2 shows this formulation in a more graphical and intuitive way.

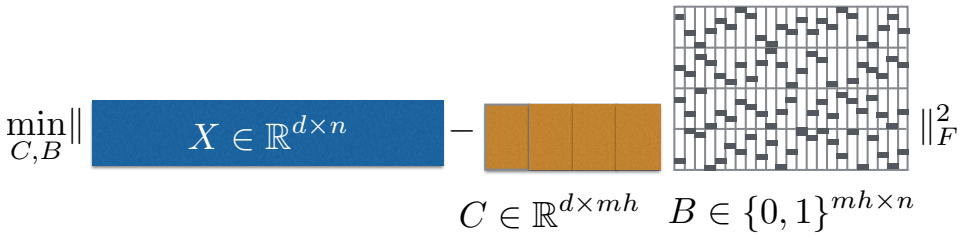

Figure 2.2: Illustration of the non-orthogonal MCQ optimization problem.

Residual vector quantization (RVQ)

The first use of non-orthogonal codebooks for ANNS is due to Chen et al. (2010). The proposed optimization approach simply optimizes the codebook sequentially using *k*-means. Their study, however, does not address the issue of the extra memory incurred by the need to store the norms of the approximations, and simply suggests storing them during the offline training stage, which makes the comparison with PQ somewhat problematic.

Enhanced residual vector quantization (ERVQ), Stacked quantizers (SQ)

Ai et al. (2014, 2017) proposed an enhanced version of RVQ, borrowing ideas from the vector quantization (VQ) literature of the 1990s (Chan et al., 1992). ERVQ uses RVQ as initialization, and further iterates to refine the learned codebooks by fixing the parameters of all but the *i*th codebook, updating its parameters by coordinate descent until convergence. The authors further propose using the short-listing approach of Hwang et al. (2012b) to speed up encoding. The same idea was independently applied to ANNS by Martinez et al. (2014), who also evaluated their approach on features obtained from deep neural networks, showing for the first time that non-orthogonal approaches produce particularly better results than orthogonal ones on such datasets.

Additive quantization (AQ)

Babenko and Lempitsky (2014a) introduced a method that has the same formulation as RVQ and ERVQ, but were apparently not aware of such previous work.

Babenko and Lempitsky put an emphasis on the combinatorial optimization problems that arise when one tries to optimize multiple codebooks jointly, and proposed an efficient least-squares formulation and method for codebook update. They also noted that finding the optimal codes after fixing the codebooks results in multiple NP-hard combinatorial problems, and proposed to use beam search for this stage. Moreover, they suggest simply quantizing the norm of the approximation and looking it up during search, which results in a query time comparable to PQ and OPQ, but achieves marginal improvements over those baselines.

Although Babenko and Lempitsky (2014a) appear to be unaware of previous work on the subject, and did not report particularly good results over PQ and OPQ, their emphasis on the computational challenges of the problem and its practical solution to the query time overhead of non-orthogonal methods have proven influential within the computer vision community.

The authors released a Python implementation of their method 11 months after publication at https://github.com/arbabenko/Quantizations.

Self-organizing binary encoding (SOBE)

Ozan et al. (2016c) proposed a method that encodes each vector greedily, similar to RVQ but using a lookahead over the next two codebooks. Moreover, after each vector is encoded, the codewords that it indexes into are updated with gradient descent. The paper presents a comparison against RVQ (Chen et al., 2010) and OCKM (Wang et al., 2014), achieving slightly better results than these two baselines.

Joint k-means quantization (JKM)

Followup work by Ozan et al. (2016b) proposed a hierarchical beam search encoding method that keeps track of a shortlist of H candidates for each codebook, and then explored the best H^2 candidates for every subsequent codebook pair. This is much faster than the beam search proposed by Babenko and Lempitsky, and achieves slightly better results than AQ (Babenko and Lempitsky, 2014a) and OTQ (Babenko and Lempitsky, 2015). Liu et al. (2016) have proposed the same encoding strategy, and called it multi-path encoding.

Competitive quantization (COMPQ)

The most recent work by Ozan et al. (2016a) uses the gradient-based codebook update of SOBE (Ozan et al., 2016c) and the hierarchical beam search encoding of JKM (Ozan et al., 2016b), and achieves better results than both methods combined, albeit without mentioning runtime.

These three papers (Ozan et al., 2016a,b,c) did not take into consideration the increased query time that comes with using full-dimensional codebooks, and were benchmarked only on datasets of SIFT and Gist (Oliva and Torralba, 2006) features.

Improving the speed of AQ encoding

Muravev et al. (2017) follow the AQ formulation and use a pyramid structure to prune beam search candidates, resulting in faster, but not always more accurate, encoding. They simply call their approach pyramid encoding. Li et al. (2017) propose finding an optimal codebook ordering for sequential encoding based on the indexing frequency of codebook indexing. They show results similar to AQ in accuracy, but with much improved runtimes. They call their approach fast AQ encoding.

MCQ *recap.* Table 2.1 shows a summary of the methods reviewed so far.

Reference	Name	Abbr.	Code
Orthogonal methods			
Brandt (2010)	Transform coding	TC	
Sandhawalia and Jégou (2010)	Expectation-based coding	–	
Jégou et al. (2009, 2011)	Product quantization	PQ	Matlab, C[2]
Ge et al. (2014)	Optimized product quantization	OPQ	Matlab
Norouzi and Fleet (2013)	Cartesian *k*-means	CKM	Matlab, C++
Douze et al. (2016)	Polysemous codes	–	C++
Semi-orthogonal methods			
Zhang et al. (2014)	Composite quantization	CQ	MSVC C++[3]
Babenko and Lempitsky (2015)	Optimized tree quantization	OTQ	
Wang et al. (2014)	Optimized Cartesian *k*-means	OCKM	
Non-orthogonal methods			
Chen et al. (2010)	Residual vector quantization	RVQ	
Ai et al. (2014, 2017)	Enhanced residual vector quantization	ERVQ	
Babenko and Lempitsky (2014a)	Additive quantization	AQ	Python[4]
Martinez et al. (2014)	Stacked quantizers	SQ	Matlab, C++
Ozan et al. (2016c)	Self-organizing binary encoding	SOBE	
Ozan et al. (2016b)	Joint *k*-means quantization	JKM	
Ozan et al. (2016a)	Competitive quantization	COMPQ	
Muravev et al. (2017)	Pyramid additive quantization	–	
Li et al. (2017)	Fast additive quantization	–	

Table 2.1: Summary of MCQ methods in the literature

[2]The Matlab version is free, and a "quite optimized" C version is available for purchase at http://people.rennes.inria.fr/Herve.Jegou/projects/ann.html. However, André et al. (2015) found that this C version is, in fact, slower than a naïve C implementation on a commodity Haswell processor.

[3]Not mentioned in the original paper. Released 37 months after publication at https://github.com/hellozting/CompositeQuantization

[4]Not mentioned in the original paper. Released 11 months after publication at https://github.com/arbabenko/Quantizations

2.4.4 Other MCQ improvements

Here we review other notable improvements to MCQ. These improvements can be applied independently from the techniques described above, and we only review them for completeness.

Quantized sparse representations

Jain et al. (2016) adapt sparse coding to the large-scale similarity search problem. In sparse coding, there is a set of (dense) codebooks, and a set of sparse, real-valued coefficients. Jain et al. further constrain the coefficients to belong to a small set. These coefficients can be learned jointly with the codebooks, and quantized with a small memory overhead. Search incurs an extra multiplication per lookup table, compared to PQ. The authors demonstrate increased accuracy over PQ and RVQ at the cost of increased query time.

Polysemous codes

Douze et al. (2016) propose polysemous codes, binary codes whose Hamming distance can be used to shortlist candidates, but whose indices are still used to index into codebooks, and thus can also be used to compute more accurate distances with lookup tables. The authors use simulated annealing to optimize their formulation, and show improved query times over PQ at the cost of a small decrease in recall.

Distance lookup caching and vectorization

Other lines of work have focused on low-level software improvements to the lookup process of MCQ.

André et al. (2015) quantize floating point lookup tables to 8 bits per value. These smaller tables fit into single instruction multiple data (SIMD) registers and accelerate the lookups necessary to compute distances in MCQ.

In parallel, Blalock and Guttag (2017) use smaller tables, and also quantize the distance tables using 8 bits. Their experiments focus on demonstrating fast encoding and the application to faster, yet approximate, matrix multiplications.

MCQ for maximum inner product search

Guo et al. (2016) propose a modification of PQ tailored for maximum inner product search (MIPS). They modify the usual k-means procedure to use the Mahalanobis distance, and demonstrate that the approximation yields an unbiased estimator. They also propose a further modification that adapts the codebooks to a particular query distribution.

Finally, Wu et al. (2017) propose multi-scale quantization, a pipeline for non-orthogonal MCQ that quantizes the norms of the database separately (similar to gain-shape quantization), and that is trainable end-to-end with gradient descent. The authors also demonstrate various theoretical guarantees on its performance, and show better empirical results compared to PQ and SQ (Martinez et al., 2014).

2.4.5 Supervised quantization

In a supervised setting, one has access to both the data and its corresponding labels. In this scenario, it is possible to use the labels to improve the accuracy of systems that compute ANNs as part of a semantic retrieval or large-scale classification pipelines. The main challenge in this area is that current deep learning systems are usually trained with backpropagation, and are thus more efficiently trained with differentiable operations. Since quantization is in general non-differentiable, work mostly focuses on optimization techniques that approximate hard quantization assignments.

The work in this thesis does not make use of labels, and thus falls under the unsupervised category. However, it is interesting to review the recent literature in supervised quantization as it provides interesting avenues for future work.

Supervised composite quantization

Wang et al. (2016) introduce a method similar to composite quantization (Zhang et al., 2014), but that has the additional constraint of maximizing the separability of different classes during encoding.

Deep quantization network

Cao et al. (2016) jointly train a neural network with a product quantizer. However,

only the network benefits from supervision, and the quantizer is updated once per-epoch without label information. Cao et al. (2017) extend this work to optimize for Maximum inner product search (MIPS).

Supervised structured binary codes (SUBIC)

More recently, Jain et al. (2017) introduced SUBIC, a neural network that both learns feature representations and produces compact codes. SUBIC manages to implicitly learn a set of codebooks by adding sparsity terms that ensure block-wise one-hot encodings, and balancing the use of all codebooks across the dataset. Klein and Wolf (2017) have further improved upon SUBIC with a network that explicitly learns soft- and hard-quantized representations. Followup work by Jain et al. (2018) introduces a differentiable large-scale indexing structure that can be learned end-to-end together with SUBIC, resulting in the first complete image indexing pipeline that can be learned end-to-end.

2.4.6 Software

While some of the papers mentioned above have released code that reproduces their results, no public library for quantization-based ANNS existed until mid-2017, when Johnson et al. (2017) released the Facebook AI Similarity Search (FAISS) library: https://github.com/facebookresearch/faiss.

The implementations in FAISS are highly optimized, written in C++ and have Python bindings readily available, which makes it easy to use and an invaluable tool in practice. Furthermore, the authors have ported most of the methods to CUDA, with additional code that automatically takes advantage of multiple GPUs in a node transparently, using a computational model similar to Tensorflow (Abadi et al., 2016). FAISS implements multiple indexing methods, such as an inverted file (Jégou et al., 2011) and the inverted multi-index (Babenko and Lempitsky, 2012), the latter being a crucial part of state-of-the-art, quantization-based systems for large-scale ANNS. However, the package only implements three vector quantization methods: PQ, OPQ and polysemous codes (Douze et al., 2016); all of them orthogonal. We believe that FAISS can benefit from implementing state-of-the-art non-orthogonal quantization methods as well.

Chapter 3

Efficient encoding and sparse codebooks

Colour my life with the chaos of trouble
—Belle and Sebastian

In multi-codebook quantization (MCQ), encoding amounts to finding, for each vector in the database, the set of assignments into the codebooks that minimize quantization error. For a given dataset and a fixed set of codebooks, the encoding problem can be expressed as maximum-a-posteriori (MAP) inference in multiple Markov random fields (MRFs) (Babenko and Lempitsky, 2014a). This process can be a computational bottleneck when MCQ is applied to large-scale databases.

The combinatorial optimization community has been dealing with similar problems for many years, and competitions to solve distribution-specific instances of NP-hard problems as efficiently as possible (e.g., the SAT competition series (Järvisalo et al., 2012)) have driven research on fast combinatorial optimization methods such as stochastic local search (SLS) (Hoos and Stützle, 2004) and portfolio-based solvers (Xu et al., 2008). Inspired by the combinatorial optimization literature, our main contribution in this chapter is the introduction of an SLS-based algorithm that achieves low quantization error at a high encoding speed in MCQ. We also discuss a series of implementation details that make our algorithm fast in practice, including a graphics processor unit (GPU) implementation. Finally, we also demonstrate an

application of our approach that incorporates sparsity constraints in the codebooks.

3.1 Background and related work

We use the same notation as introduced in Chapter 2. Formally, we denote the set to quantize as $X \in \mathbb{R}^{d \times n}$, having n data points with d dimensions each; MCQ is the problem of finding m codebooks $C_i \in \mathbb{R}^{d \times h}$ and the corresponding codes B_i that minimize quantization error, *i.e.*, to determine

$$\min_{C_i, B_i} \|X - [C_1, C_2, \ldots, C_m] \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_m \end{bmatrix} \|_F^2, \tag{3.1}$$

where $B_i = [\mathbf{b}_{i1}, \mathbf{b}_{i2}, \ldots, \mathbf{b}_{in}] \in \{0,1\}^{h \times n}$, and each subcode $\mathbf{b}_{ij}$ is limited to having only one non-zero entry: $\|\mathbf{b}_{ij}\|_0 = 1$, $\|\mathbf{b}_{ij}\|_1 = 1$. Letting $C = [C_1, C_2, \ldots, C_m]$ and $B = \left[B_1^\top, B_2^\top, \ldots, B_m^\top\right]^\top$, we can rewrite expression 3.1 more succinctly as

$$\min_{C,B} \|X - CB\|_F^2. \tag{3.2}$$

Although Babenko and Lempitsky were not technically the first to use this formulation (that credit goes to Chen et al. (2010)), their paper was the first to propose a non-greedy optimization approach based on block coordinate descent ("EM-like"), consisting of, first, initializing B randomly, and then

- updating C while maintaining B fixed, and
- updating B while maintaining C fixed,

until convergence. In this approach, updating the codebooks C amounts to solving a large-scale least-squares problem, and updating B results in n hard combinatorial optimization problems. In this chapter we focus on the latter problem.

3.1.1 The encoding problem in MCQ

Our work focuses on improving the encoding time and performance of block coordinate approach approaches to MCQ – that is, given the data X and codebooks C,

we search for a method to find the codes B that minimize Expression 3.2.

Formally, the encoding problem amounts to maximum-a-posteriori (MAP) inference[1] in n fully-connected MRFs (one for each item in X) with m nodes each (one per subcode). These MRF nodes may take any value between 1 and h, so in general they are not binary. In the general case, this problem is NP-hard.

In these MRFs, the minimum energy is achieved by finding the subcodes $\mathbf{b}_i$ that minimize the squared distance between the vector to encode $\mathbf{x}$, and its approximation $\hat{\mathbf{x}}$:

$$\|\mathbf{x}-\hat{\mathbf{x}}\|_2^2 = \|\mathbf{x}-\sum_i^m C_i\mathbf{b}_i\|_2^2 = \|\mathbf{x}\|_2^2 - 2\cdot\sum_i^m\langle\mathbf{x}, C_i\mathbf{b}_i\rangle + \|\sum_i^m C_i\mathbf{b}_i\|_2^2 \tag{3.3}$$

where the norm of the approximation $\|\hat{\mathbf{x}}\|_2^2 = \|\sum_i^m C_i\mathbf{b}_i\|_2^2$ can be expanded as

$$\|\sum_i^m C_i\mathbf{b}_i\|_2^2 = \sum_i^m\|C_i\mathbf{b}_i\|_2^2 + \sum_i^m\sum_{j\neq i}^m\langle C_i\mathbf{b}_i, C_j\mathbf{b}_j\rangle. \tag{3.4}$$

Posed as an MRF with m nodes, the $\|\mathbf{x}\|_2^2$ term in (3.3) can be discarded because it is a constant with respect to $\mathbf{b}_i$; the $-2\cdot\sum_i^m\langle\mathbf{x}, C_i\mathbf{b}_i\rangle$ and $\sum_i^m\|C_i\mathbf{b}_i\|_2^2$ terms are summed up and become the unary terms, and $\sum_i^m\sum_{j\neq i}^m\langle C_i\mathbf{b}_i, C_j\mathbf{b}_j\rangle$ yields the pairwise terms. Since each code $\mathbf{b}_i$ may take any value between 1 and h, there are h^m possible configurations for each MRF (typically, $m = \{8, 16\}$, and $h = 256$), which renders the problem inherently combinatorial and, in general, NP-hard (Cooper, 1990).

On the empirical challenges of AQ-encoding

Solving the MRFs that arise in AQ encoding is known to be challenging; in fact, Babenko and Lempitsky (2014a) noted:

> The optimization problem can be solved approximately by any of the existing algorithms like loopy belief propagation (LBP), iterated conditional modes (ICM), etc. However, because of full connectivity and the general form of the pairwise potentials, we observed that LBP and

[1] Unfortunately enough, Maximum-a-posteriori (MAP) inference is often called "decoding", due to its historical use in the receiving end of error-correcting codes (Gersho and Gray, 1992).

> ICM, and, in fact other algorithms from the MRF optimization library [Kappes et al. (2013)] perform poorly.

Babenko and Lempitsky (2014a) proposed resorting to beam search, a computationally expensive construction search method. This method, however, achieved only marginal improvements over the baselines PQ an OPQ. Moreover, the authors also note:

> Interestingly, we tried to formulate the MRF optimization [...] as an integer quadratic program and submit it to a general purpose branch-and-cut optimizer [(Gurobi)]. For small codebooks $[h] = 64$ and $[m] = 8$ and given very large amount of time, the solver could find a global minimum with much smaller energy (coding error) then [sic] those found by [b]eam [s]earch or other non-exhasutive [sic] algorithms. While such "smart bruteforce" approach is infeasible for large datasets and meaningful codebook sizes, this result suggests the AQ-coding problems as interesting instances for the [sic] research into new MRF-optimization algorithms.

We build on this observation, and look for an encoding algorithm with improved computation vs accuracy trade-offs for this problem.

3.1.2 Reducing query time in AQ

AQ (as well as most other non-orthogonal MCQ techniques) involves major computational overhead during query time. The overhead occurs when a new query $\mathbf{q}$ is received, and we estimate the distance to the encoded vectors $\hat{\mathbf{x}}_i$. This amounts to evaluating

$$\|\mathbf{q} - \hat{\mathbf{x}}\|_2^2 = \|\mathbf{q} - \sum_i^m C_i \mathbf{b}_i\|_2^2 = \|\mathbf{q}\|_2^2 - 2 \cdot \sum_i^m \langle \mathbf{q}, C_i \mathbf{b}_i \rangle + \|\sum_i^m C_i \mathbf{b}_i\|_2^2 \tag{3.5}$$

which requires $O(m^2)$ table lookups for evaluating the norm of the encoded vector $\|\hat{\mathbf{x}}\|_2^2$.

Babenko and Lempitsky (2014a) proposed a simple solution to this problem: use $m-1$ codebooks to quantize $\mathbf{x}$, and use an extra byte to quantize the norm of

the approximation $\|\hat{\mathbf{x}}\|_2^2$. This approximation is likely to be very good, as we are then using $h = 256$ centroids to quantize a scalar. Using this extra table is crucial to make sure that the query time of non-orthogonal MCQ methods is comparable with PQ and OPQ. Unfortunately, with this constraint in mind, the improvement in recall of AQ over PQ and OPQ became rather marginal (see AQ-7 in Figure 5 of Babenko and Lempitsky (2014a)).

We use $m-1$ codebooks in all our experiments, which makes sure that our query time is comparable with previous work.

3.1.3 MAP estimation of MRFs using GPUs

Our broader goal is to come up with algorithms that scale well on large datasets, so we want our algorithm to be fast in practice. Since the encoding problem decomposes into n independent MRFs, we also investigate the use of parallel architectures – in particular GPUs – for accelerating MAP estimation in MRFs.

MRFs are a widely-used tool in computer vision, frequently leveraged to model visual smoothness in problems such as stereo, semantic segmentation, optical flow and visual localization. The first connection between these statistical models and images is due to Geman and Geman (1984). In this context, Boykov et al. (2001) propose graph cuts, an iterative algorithm that makes large moves in labelling over multiple nodes at once, achieving fast convergence. Krähenbühl and Koltun (2011) use filtering to perform MAP inference on MRFs with binary terms coming from Gaussian kernels, and apply their optimization method to semantically segment natural images.

Given the widespread use of MRFs as a modelling tool in computer vision, we have found previous work that performs MAP estimation in the GPU. Vineet and Narayanan (2008) introduce CUDACuts, a CUDA-based implementation of the graph cuts algorithm by Boykov et al., which achieves exact Markov random field (MRF) optimization on submodular MRFs. The implementation exploits shared memory in the push-relabel step of graph cuts, and is demonstrated on a Nvidia GTX 280 GPU. Zach et al. (2008) introduce a data-parallel approach to MRF optimization based on plane sweeping, which is applicable when the pairwise terms follow a Potts prior. The work dates back to 2008, when CUDA was a more

difficult language to program in than it is today, so the authors used the OpenGL application programming interface (API) to program their solution. The authors demonstrate an implementation on an Nvidia Geforce 8800 Ultra GPU.

Most previous work focuses on MRFs applied to image-related tasks, such as scene segmentation. In these applications, the MRFs typically have a large number of nodes (1 per pixel), a low number of labels (e.g., 3 (Zach et al., 2008)), are sparsely connected, and have special constraints on their pairwise terms (e.g. Potts in the work of Zach et al. (2008) and submodular in the work of Vineet and Narayanan (2008)). Thus, it is unlikely that those implementations will fit our problem, which has a low number of nodes $m = \{8, 16\}$, a relatively large number of labels ($h = 256$), is densely connected, and whose pairwise terms are not constrained to follow any particular formulation.

3.1.4 Stochastic local search (SLS) algorithms

Top-performing methods for solving many NP-hard problems have been, at several points in time, variations of stochastic local search (SLS), and continue to define the state of the art for solving prominent NP-hard problems, such as the travelling salesperson problem (TSP) (Nagata and Kobayashi, 2013). Given a candidate solution to a given problem instance, SLS methods iteratively examine neighbouring candidates, and replace the candidate solution with one of its neighbours until a stop criterion is met. A formal treatment of the subject involves defining neighbourhood functions, characterizing problem instances and formally defining local search procedures; while this is beyond the scope of this work, we direct interested readers to the book by Hoos and Stützle (2004).

Iterated local search (ILS)

Iterated local search (ILS) algorithms (Lourenço et al., 2003), which form the basis for the algorithm we propose in this work, alternate between perturbing the current solution s (with the goal of escaping local minima) and performing local search starting from the perturbed solution, leading to a new candidate solution, s'. At the end of each local search phase, a so-called acceptance criterion is used to decide whether to continue the search process from s or s'. In many applications of ILS,

including ours described in the following, the acceptance criterion simply selects the better of s and s'.

Theoretical bounds of SLS algorithms

A downside of SLS algorithms (and many other heuristic methods that perform well in practice) is that their theoretical performance has historically proven hard to analyze. Similar to prominent deep learning techniques, SLS methods often perform far better than theory predicts, and thus, research in the area is heavily based on empirical analysis. An attempt to achieve a theoretical breakthrough for our method would be out of the scope of this work, so instead, we focus on the thorough empirical evaluation of its performance on a number of benchmarks with varying sizes, protocols and descriptor types to show the strength of our approach.

3.2 Iterated local search for MCQ encoding

We now introduce our ILS algorithm to optimize B in Expression 3.2. Our algorithm is defined by

1. an initialization procedure (to create the first solution)
2. a perturbation procedure (to escape local minima)
3. a local search algorithm, and
4. an acceptance criterion (stated above).

We now explain our design choices for these four components.

3.2.1 Local search procedure

As our local search procedure we choose cyclic iterated conditional modes ICM. Iterated conditional modes ICM offers two key advantages over other local search algorithms: (i) on the theoretical side, it exhibits very good speed, and, (ii) in practice, it can be implemented in a way that is amenable to caching and vectorization. Although ICM by itself does not tend to perform well on large MRF models applied to full images, our problem is different enough that we consider it a suitable candidate. We discuss both advantages in more detail below.

Complexity analysis of ICM

Iterated conditional modes ICM iterates over all the nodes in the given MRF, conditioning the current node on the assignments of other nodes, and minimizing (finding the mode of) the resulting univariate distribution. In a fully-connected MRF, such as the one in our problem, each node represents a subcode $\mathbf{b}_i$. Thus, ICM cycles through m subcodes, and conditions its value on the other $m-1$ subcodes, adding h terms for each conditioning. This results in a complexity of $O(m^2 \cdot h)$. Comparing to the beam search procedure of AQ, which has a complexity of $O(m \cdot h^2(m + \log m \cdot h))$ (Babenko and Lempitsky, 2015), we can see that for typical values of $m \in \{8, 16\}$ and $h = 256$, a single ICM cycle is much faster than beam search. This suggests that we can afford to run several rounds of ICM, which is necessary for ILS, while keeping the overall encoding time low. In practice, our implementation is 30-50 times faster than the beam search implementation due to Babenko: https://github.com/arbabenko/Quantizations.

Cache hits and vectorization of ICM

While this is not true in general, in our special case of interest ICM has a second crucial advantage: it can be programmed in a way that is cache-friendly and easy to vectorize.

In practice, the computational bottleneck of ICM arises in the conditioning step, when the algorithm looks at all the neighbours of the ith node and adds the assignments in those nodes to the current one. A naïve implementation, such as that available from off-the-shelf MRF libraries (Kappes et al., 2013), encodes each data point sequentially, looking up the pairwise terms from $O(m^2)$ different tables of size $h \times h$. This results in a large number of cache-misses, as different tables are loaded into cache for each conditioning.

Our observation from Equation 3.3 is that only the unary term $-2\sum_i^m \langle \mathbf{x}, C_i \mathbf{b}_i \rangle$ depends on the vector to encode $\mathbf{x}$. We also note that the pairwise terms $\sum_i^m \sum_{j \neq i}^m \langle C_i \mathbf{b}_i, C_j \mathbf{b}_j \rangle$ are the same for all the MRFs that arise in the encoding problem. This means that, during the conditioning step, we can condition all the ith subcodes in the database w.r.t. the jth subcode, loading only one $h \times h$ pairwise lookup table into cache at a time. This results in better caching performance, is easily vectorized, and dramati-

cally speeds up ICM in our case. In practice, this is accomplished by switching the order of the for loops in ICM over the entire (or a large portion of the) database. We refer to this implementation as "batched Iterated conditional modes (ICM)".

Other variants of ICM

Other local search procedures may be used instead of cyclic ICM. For example, one may explore all the combinations of multiple codes at a time and update them jointly (block ICM), or update only the code that offers the best improvement (greedy ICM). These algorithms are all implemented in the UGM toolbox due toSchmidt (2007). Greedy ICM may be implemented efficiently using a heap (Nutini et al., 2015, Appendix. A). Exploring the cost and empirical performance of this implementation is an interesting area of future work.

3.2.2 ILS Perturbation

In each incumbent solution s, we choose k codes to perturb by sampling without replacement from the discrete uniform distribution from 1 to m: $i_k \sim \mathscr{U}\{1,m\}$. We then perturb each selected code uniformly at random by setting it to a value between 1 and h: $\mathbf{b}_{i_k} \sim \mathscr{U}\{1,h\}$. This perturbed solution s' is then used as the starting point for the subsequent local search phase, i.e., invocation of ICM. While simple, this perturbation method is commonly used in the SLS literature (Hoos and Stützle, 2004). We note that our approach generalizes previous work where ICM was used but no perturbations were applied (Zhang et al., 2014), which corresponds to setting $k = 0$. This method is both effective and very fast in practice: compared to ICM, the time spent in this step is negligible.

3.2.3 ILS Initialization

We use a simple initialization method, setting all the codes to values between 1 and h uniformly at random: $\mathbf{b}_i \sim \mathscr{U}\{1,h\}\ \forall i \in \{1,2,\ldots,m\}$. We also experimented with other initialization approaches, such as using FLANN (Muja and Lowe, 2009) to copy the codes of the nearest neighbour in the training dataset, or using the code that minimizes the binary terms (which are expected to dominate the unary terms for large m). However, we did not observe significant improvements over

our random initialization after a few rounds of ILS optimization.

Overall initialization

Like AQ and composite quantization (CQ), we use an auxiliary quantization method to initialize B and C in our optimization procedure. We run OPQ, followed by a method similar to optimized tree quantization (OTQ) (Babenko and Lempitsky, 2015), but simplified to assume that the dimension assignments are given by a natural partition of adjacent dimensions.

3.2.4 Pseudocode

We now provide the pseudocode for our algorihtm.

3.3 Graphics processor unit (GPU) implementation

GPUs are computing devices originally conceived for graphics applications, capable of performing the same operation over many data points in a single clock cycle, achieving higher throughput than CPUs. Thus, they are a good fit for problems that are amenable to parallelization. Next, we describe the details of our GPU implementation.

Our implementation consists of three main CUDA kernels: (a) a setup kernel, which initializes the solutions, computes unary lookup tables and creates a series of random number generators; (b) a perturbation kernel, which alters random codes in the solution, and (c) a local search kernel, which implements ICM.

Initialization kernel

Our initialization kernel creates a CUDA random (`curand`) state for each data point in the database and, during the first iteration, uses the samples to create a random initial solution. The implementation takes less than $1/1\,000$ of the total running time.

Algorithm 1 Our SLS algorithm for encoding in MCQ

function ENCODE(
  $\mathbf{x} \in \mathbb{R}^d$, ▷ Vector to encode
  $\mathbf{b} \in \{0,1\}^{m \times h} = [\mathbf{b}_i, \mathbf{b}_2, \ldots, \mathbf{b}_m]; \mathbf{b}_i \in \{0,1\}^h$, ▷ Initial codes
  $k \in \{0,1,\ldots,m\}$, ▷ Number of codes to perturb
  $m \in \mathbb{Z}$, $h \in \mathbb{Z}$, ▷ Number of codebooks and entries in each codebook
  $I \in \mathbb{Z}$, $J \in \mathbb{Z}$) ▷ Number of ILS and ICM iterations
  for $i \leftarrow 1,2,\ldots,m$ **do**
    $\mathbf{b}_i \sim onehot(\mathscr{U}\{1,h\})$ ▷ Initialize each code to a random value.
  end for
  $s \leftarrow [\mathbf{b}_i, \mathbf{b}_2, \ldots, \mathbf{b}_m]$ ▷ Save the current solution
  for $i \leftarrow 1,2,\ldots,I$ **do** ▷ Loop over ILS iterations.
    $\mathbf{b} \leftarrow$ PERTURB$(\mathbf{b}, k, m)$ ▷ Perturb k codes
    for $j \leftarrow 1,2,\ldots,J$ **do** ▷ Loop for the number of ICM iterations
      for $k \leftarrow 1,2,\ldots,m$ **do** ▷ Update each code, keeping the rest fixed
        $unary \leftarrow -2\langle \mathbf{x}, C_k \cdot \mathbf{b}_k \rangle + \|C_k \cdot \mathbf{b}_k\|_2^2$
        $binary \leftarrow \sum_{i \neq k}^{m} \langle C_k \cdot \mathbf{b}_k, C_i \cdot \mathbf{b}_i \rangle$
        $\mathbf{b}_k \in \text{argmin}_{\mathbf{b}_k}(unary + binary)$
      end for
    end for
    $s' \leftarrow [\mathbf{b}_i, \mathbf{b}_2, \ldots, \mathbf{b}_m]$
    if $cost(s') < cost(s)$ **then**
      $s \leftarrow s'$ ▷ If the new solution is better, keep it
    end if
  end for
  return s
end function

Perturbation kernel

In this kernel, we have one thread per data point, with the goal of maximizing the work/thread ratio. The task of each thread is to choose k entries in the current solution, and perturb them to random values between 1 and h.

The main challenge in this kernel is that the choice of k entries to perturb amounts to sampling without replacement from the uniform distribution between 1 and m. Since there are no off-the-shelf functions for sampling without replacement

in the `curand` library, we implemented our own version of *reservoir sampling*[2]. The pseudocode for this kernel is shown as Algorithm 2. Here we show our perturbation algorithm in pseudocode.

Algorithm 2 Our perturbation method using reservoir sampling

function PERTURB(
  $\mathbf{b} \in \{0,1\}^{m \times h} = [\mathbf{b}_i, \mathbf{b}_2, \ldots, \mathbf{b}_m]; \mathbf{b}_i \in \{0,1\}^h$, ▷ Initial codes
  $k \in \{0,1,\ldots,m\}$, ▷ Number of codes to perturb
  $m \in \mathbb{Z}$, $h \in \mathbb{Z}$) ▷ Number of codebooks and entries in each codebook
  $need \leftarrow k$ ▷ The number of codes that we still have to perturb
  $left \leftarrow m$ ▷ The number of codes that we can still visit
  for $i \leftarrow 1,2,\ldots,m$ **do**
    $r \sim \mathscr{U}(0,1)$ ▷ Sample a value between 0 and 1
    if $r < need/left$ **then** ▷ With probability $(need/left)$...
      $\mathbf{b}_i \sim \mathscr{U}\{1,h\}$ ▷ Alter the ith code
      $need \leftarrow need - 1$ ▷ Decrease the codes we have left to perturb
      if $need \leq 0$ **then**
        return b ▷ If we have perturbed enough codes, we are done
      end if
    end if
    $left \leftarrow left - 1$ ▷ Decrease the codes we can still perturb
  end for
  return b ▷ Return the perturbed code
end function

It is easy to show by induction that each code has an equal probability k/m of being perturbed, and the algorithm finishes when k samples have been perturbed – which is most likely achieved before visiting every code. Although elegant, reservoir sampling leads to large thread divergence, which is likely suboptimal for the GPU architecture. Fortunately, the time spent in this perturbation step is negligible compared to the ICM kernel that we discuss next.

[2] https://en.wikipedia.org/wiki/Reservoir_sampling

ICM kernel

The task of the ICM kernel is to perform local search after the solution has been perturbed; this corresponds to lines 8-14 in Algorithm 1. This is, by far, the most expensive part of our pipeline, so we discuss the implementation in more detail.

In short, we have three main tasks: (a) copying the pre-computed unary terms to shared memory, (b) adding the corresponding pairwise terms to the unary terms for each possible value of each code, and (c) finding the minimum of all h possibilities in each code.

Task (a) is only done once and its implementation is straightforward. Task (b) is memory-limited – the pairwise tables are too big to fit in shared memory, so they have to be read from global memory. Task (c) amounts to a reduce operation, and is compute-limited.

ICM *conditioning.* The main challenge in ICM conditioning is to access the pairwise terms in a coalesced manner.

We simply store each table of pairwise terms

$$T_{i,j} = -2 \cdot C_i^\top \cdot C_j, \;\; T_{i,j} \in \mathbb{R}^{h \times h}. \tag{3.6}$$

Since $T_{i,j} = T_{j,i}^\top$, this incurs extra memory usage, but allows for a simple implementation where the i, j represent the code being minimized over (i), and the code whose terms are being accumulated (j). Once the corresponding table is loaded into cache, we can simply use the current codes to add the corresponding pairwise terms, thus preserving coalesced memory access.

Reduce operation. Once all the pairwise terms have been added, we run task (c) as a reduce `findmin` operation on the sum of unary and pairwise terms. This is a reduction over the h values that we computed in task (b). Note, however, that the output is not the minimum value itself, but its index (a number from 1 to h), so we use an extra array of indices during reduction. We implemented all the optimizations available in the CUDA reduce guide (https://docs.nvidia.com/cuda/samples/6_Advanced/reduction/doc/reduction.pdf) for this step.

3.4 The advantages of a simple formulation: easy sparse codebooks

Our approach, building on top of AQ, benefits from using a simple formulation with no extra constraints (Expression 2.4). The advantages of a simple formulation are not merely aesthetic; in practice, they result in a straightforward optimization procedure and less overhead for the programmer. Furthermore, a more standard formulation might render other variants of the problem easier to solve as well. We now demonstrate one such use case, by implementing a recent MCQ formulation that enforces sparsity on the codebooks (Zhang et al., 2015).

Motivation for sparse codebooks

Zhang et al. (2015) motivate the use of sparse codebooks in the context of very large-scale applications, which deal with billions of data-points and are better suited for search with an inverted-index (Babenko and Lempitsky, 2012, 2014b; Xia et al., 2013). In this case, the time spent computing the lookup tables becomes non-negligible, which is specially true for methods that use full-dimensional codebooks, such as CQ (Zhang et al., 2014), AQ (Babenko and Lempitsky, 2014a) and, by extension, ours. For example, Zhang et al. (2015) have demonstrated that enforcing sparsity in the codebooks can lead up to 30% speedups with an inverted index (Babenko and Lempitsky, 2014b) on the SIFT1B dataset (see Table 2 in (Zhang et al., 2015)). This method is called sparse composite quantization (SCQ).

There is a second use case for sparse codebooks. Recently, André *et al* (André et al., 2015) have demonstrated that PQ scan and other lookup-based sums, such as those in our approach, can take advantage of vectorization. The authors have demonstrated speedups of up to a factor of 6 on distance computation, thus emphasizing further the time spent computing distance tables even for datasets with a few million data points, where linear scanning is the preferred search procedure.

The sparsity constraint

Formally, the sparsity constraint on the codebooks modifies the typical MCQ formulation such that, in this case, we want to determine

$$\min_{C,B} \|X - CB\|_F^2 \quad \text{s.t.} \quad \|C\|_0 \leq S. \tag{3.7}$$

Unfortunately, minimizing a quadratic function with an ℓ_0 constraint is non-convex and, in general, hard to solve directly. Thus, the problem is often relaxed the minimize the convex ℓ_1 norm. Our objective then becomes to determine

$$\min_{C,B} \|X - CB\|_F^2 \quad \text{s.t.} \quad \|C\|_{1,1} \leq \lambda. \tag{3.8}$$

In SCQ (Zhang et al., 2015), the problem becomes even harder, because on top of sparsity, the codebook elements are forced to have constant products. For this reason, Sparse composite quantization (SCQ) uses an ad-hoc iterative soft-thresholding algorithm to find C. Our problem is simpler, because we do not have to satisfy the inter-codebook constraint and can be posed as an ℓ_1-regularized least-squares problem. To achieve this, we rewrite Expression 3.8 as

$$\min_{C,B} \|B^\top C^\top - X^\top\|_F^2 \quad \text{s.t.} \quad \|C\|_{1,1} \leq \lambda, \tag{3.9}$$

and it becomes apparent that the approximation of the ith column of $X^\top$ depends only on product of $B^\top$ and the ith column of $C^\top$. Thus, we can rewrite Expression 3.9 as

$$\min_{\hat{\mathbf{c}},B} \|\hat{B}\hat{\mathbf{c}} - \hat{\mathbf{x}}\|_2^2 \quad \text{s.t.} \quad \|\hat{\mathbf{c}}\|_1 \leq \lambda, \tag{3.10}$$

where

$$\hat{B} = \begin{bmatrix} B_{(1)}^\top & 0 & \dots & 0 \\ 0 & B_{(2)}^\top & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & B_{(d)}^\top \end{bmatrix}, \hat{\mathbf{c}} = \begin{bmatrix} \mathbf{c}_1^\top \\ \mathbf{c}_2^\top \\ \vdots \\ \mathbf{c}_d^\top \end{bmatrix}, \text{ and } \hat{\mathbf{x}} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_d^\top \end{bmatrix}. \tag{3.11}$$

Here, $B_{(i)}^\top$ is the ith copy of $B^\top$, $\mathbf{c}_i$ is the ith row of C, and $\mathbf{x}_i$ is the ith row of X.

To update $\hat{B}$ we use our previously introduced ILS procedure. Updating $\hat{\mathbf{c}}$, assuming $\hat{B}$ and $\hat{\mathbf{x}}$ are fixed, corresponds to the well-known lasso (Tibshirani, 1996). Nearly two decades of research on the lasso have produced many robust and scalable off-the-shelf optimization routines for this problem.

Our approach, as opposed to previous work (Zhang et al., 2015), lacks extra constraints and thus can directly take advantage of lasso optimizers with little overhead to the programmer. For example, it took us only a few hours to integrate the Spectral projected gradient (SPGL1) solver by van den Berg and Friedlander (2008) into our pipeline. This solver has the additional advantage of not requiring an explicit representation of $\hat{B}$, but can instead be given a function that evaluates to $\hat{B}\hat{\mathbf{c}}$ – this can be implemented as a `for` loop, so we only need to store one copy of the codes $B^\top$ in memory.

SPGL1 thresholding

Expression 3.9 requires setting the value of λ. We follow a block coordinate descent procedure, and alternate between optimizing C (using SPGL1) and B (using ILS), and set $\lambda = \alpha \cdot \tau$, where τ is the ℓ_1 norm of the solution given by PQ, which we also use for initialization.

For a given set of $\lambda = \alpha \cdot \tau$, SPGL1 may not return a solution with the level of sparsity desired; thus, at the end of each SPGL1 iteration we keep only the entries in C with the largest S absolute values, and set everything else in C to zero. We perform a grid search for the best value of $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$, and choose the value that gives the lowest quantization error on the training set.

3.5 Experiments

We now describe the experimental setup that empirically validates the impact of our contributions.

3.5.1 Evaluation protocol

We follow previous work and evaluate the performance of our system with recall@N (Babenko and Lempitsky, 2014a, 2015; Ge et al., 2014; Jégou et al., 2011; Zhang et al., 2014). Recall@N is a monotonically increasing curve from 1 to N

representing the empirical probability, computed over the query set, that the N estimated nearest neighbours include the actual nearest neighbour in the database. The goal is to obtain the highest possible recall for given N. In information retrieval, recall@1 is considered the most important number on this curve. Also in line with previous work (Babenko and Lempitsky, 2014a, 2015; Ge et al., 2014; Jégou et al., 2011; Zhang et al., 2014, 2015), in all our experiments, the codebooks have $h = 256$ elements, so we use 8 bits of memory per code. We show results using 64 and 128 bits for code length.

We compute approximate squared Euclidean distances applying the expansion of Equation 3.3, and we use only 7 and 15 codebooks to store codebook indices, while the last 8 bits are dedicated to quantize the (scalar) squared norm of each database entry. In all our experiments, we run our method for 100 iterations and use asymmetric distance (Jégou et al., 2011), i.e., the distance tables are computed for each query, as in all our baselines.

3.5.2 Baselines

We compare our approach to previous work, controlling for two critical factors in large-scale retrieval: code length and query time. To control for code length, we use subcodebooks with $h = 256$ entries and produce final codes of 64 and 128 bits. To control for query time, we restrict our comparison to methods that require $m = \{8, 16\}$ table lookups to compute approximate distances. Thus, we compare against PQ (Jégou et al., 2011), OPQ (Ge et al., 2014), RVQ (Chen et al., 2010), ERVQ/SQ (Ai et al., 2014; Martinez et al., 2014) and CQ (Zhang et al., 2014), as well as the AQ-7 method introduced by Babenko and Lempitsky (2014a) (i.e., the original formulation of AQ that we are building on).

We use our own implementations of all these methods, except for CQ. We use the public CQ implementation due to Zhang,[3] with minor modifications to run on a Linux system, since the original implementation contains C++ constructs exclusive to Windows systems. We also take the results reported in the original paper introducing AQ (Babenko and Lempitsky, 2014a). We compare our sparse extension against the values reported for SCQ by Zhang et al. (2015). To the best of our

[3] https://github.com/hellozting/CompositeQuantization

knowledge, the paper by Zhang et al. is the only one on the subject, and the authors have not released code to reproduce their results.

3.5.3 Datasets

We test our approach on 6 datasets. Three of these, SIFT1M, SIFT10M and SIFT1B (Jégou et al., 2011), consist of 128-dimensional gradient-based, hand-crafted SIFT features. We also collected a dataset of 128-dimensional deep features using the CNN-M-128 network provided by Chatfield et al. (2014), computing the features from a central 224×224 crop of the 1.2M images of the ImageNet Large-Scale Visual Recognition Challenge (ILSRVC) 2012 dataset, and then subsampling uniformly at random from all classes. It is known that deep features can be effectively used as descriptors for image retrieval (Babenko et al., 2014; Razavian et al., 2014), and Chatfield et al. have shown that this intra-net compression results in a minimal loss of retrieval performance.

SIFT1M and Convnet1M both have 100 000 training vectors, 10 000 query vectors and 1 million database vectors. SIFT1B has 100 million vectors for training, and 1 billion vectors for base, as well as 10 000 queries. On SIFT1B, we follow previous work (Ge et al., 2014; Zhang et al., 2014) and used only the first 1 million vectors for training. Better results on SIFT1B can be obtained using an inverted index, but we did not implement this data structure as we focus on improving encoding performance. This also has the added benefit of making our results directly comparable to those shown in CQ (Zhang et al., 2014) and CKM (Norouzi and Fleet, 2013). With SIFT10M, we followed the same protocol, but use only the 10 million first vectors of SIFT1B as base.

We also use 2 datasets where CQ was benchmarked: MNIST (LeCun et al., 1998) and LabelMe22K (Norouzi and Fleet, 2011). MNIST is 784-dimensional, and has 10 000 vectors for query and 60 000 vectors for base. LabelMe22K is 512-dimensional and has 2 000 vectors for query and 20 019 vectors for base.

Protocols

Different datasets have different partitions, and this leads to important differences in the way learning and encoding are performed. The classical datasets SIFT1M,

SIFT10M and SIFT1B (Jégou et al., 2011), as well as our Convnet1M dataset have three partitions: train, query and database. In this case, the standard protocol is to first learn the codebooks using only the train set, then use the learned codebooks to encode the database, and finally evaluate recall@N of the queries with respect to the database (Babenko and Lempitsky, 2014a, 2015; Ge et al., 2014; Jégou et al., 2011; Ozan et al., 2016a; Zhang et al., 2015). Zhang et al. (2014), however, used a different protocol for the results reported on their paper. In our experiments, we run their code using the standard protocol. We refer to this partition as *train/-query/base*.

Some datasets, however, have only two partitions of the data: query and database. In this case, the iterative codebook learning procedure is run directly on the database, and recall@N is evaluated on the queries with respect to the database thereafter. For example, Locally Optimized PQ (Kalantidis and Avrithis, 2014) was evaluated using this partition on SIFT1B by simply ignoring the training set, and CQ was evaluated on MNIST and LabelMe22K using the train/test partitions that the datasets provide for classification (Zhang et al., 2014). It has also been argued that this partition is better suited for learning inverted indices on very large-scale datasets (see the last paragraph of (Babenko and Lempitsky, 2014b)). We refer to this partition as *query/base*.

3.5.4 Parameter settings

Our approach needs to set the number of ILS iterations (i.e., the number of times a solution is perturbed and local search is done). At the same time, ICM may cycle through the nodes a number of times, which we call *ICM iterations*. Finally k, the number of elements to perturb, also needs to be defined.

We chose our parameters using a subset of the training set of SIFT1M, keeping the values that minimize quantization error. Figure 3.1 shows the results of our parameter search on 10 000 SIFT descriptors. We note that, given the same number of ICM cycles, using 4 ICM iterations and perturbing $k = 4$ code elements results in good performance. We observed similar results on other descriptor types, so we used these values in all our experiments. As shown in Figure 3.1, the performance of our system depends on the number of ILS iterations used, trading-off compu-

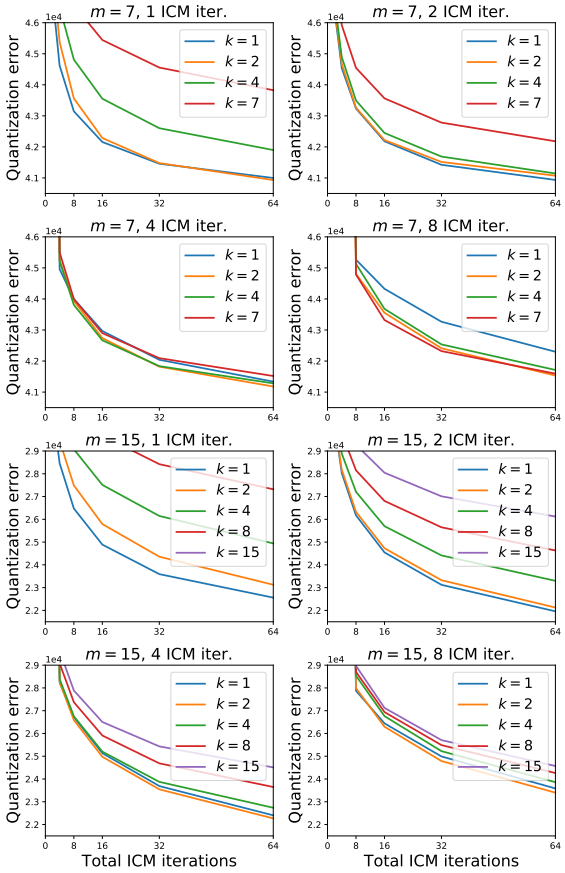

Figure 3.1: Quantization error as a function of ILS iterations, ICM iterations and number of codes perturbed k on 10 000 vectors of the SIFT1M train dataset after random initialization. Using 4 perturbations and 4 ICM iterations gives good results in all cases, so we use those parameters in all our experiments. Using no perturbations ($k = 0$), as done in previous work (Babenko and Lempitsky, 2014a; Zhang et al., 2014), leads to values that are above all the plots that we are showing, and stagnates after about 3 ICM iterations.

tation for recall. We evaluated our method using 16 and 32 ILS iterations on the base set of train/query/base datasets, and refer to these methods as Local search quantization LSQ-$\{16, 32\}$. During training, we used only 8 ILS iterations.

3.6 Results

Since our method relies heavily on randomness for encoding, it is natural to think that the final performance of the system could exhibit large variance. To quantify this effect, we ran our method 10 times on each dataset, and report the mean and standard deviation in recall@N achieved by our method. To see how this compares to previous work, we ran all the baselines 10 times and report their mean and standard deviation in recall@N as well. We observe that Local search quantization (LSQ), despite relying heavily on randomness for encoding, exhibits a variability in recall similar to that of PQ and OPQ (and presumably that of other baselines as well).

This is consistent with the fact that LSQ solves a large number of independent instances of combinatorial optimization problems of similar difficulty; in situations like this, the solution qualities achieved within a fixed running time are typically normal-distributed (Hoos and Stützle, 2004, Chapter 4). In other words, despite being heavily randomized, the performance of our system turns out to be very stable in practice, because it is averaged over a large number of data points.

3.6.1 Small training/query/base datasets

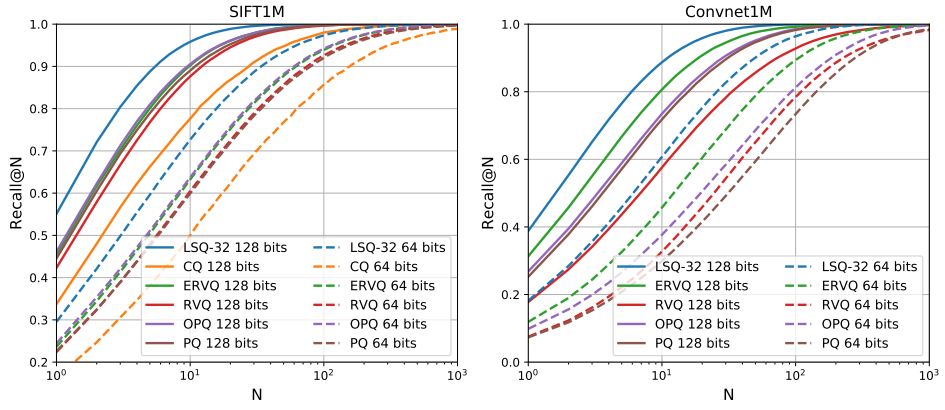

Figure 3.2: Recall@N curves for (left) the SIFT1M, and (right) Convnet1M datasets.

First, we report the recall@N results for SIFT1M and Convnet1M in Figure 3.2,

	SIFT1M – 64 bits			SIFT1M – 128 bits		
	R@1	R@10	R@100	R@1	R@2	R@5
PQ	22.59 ± 0.43	59.90 ± 0.28	91.98 ± 0.17	44.53 ± 0.40	60.49 ± 0.40	78.94 ± 0.28
OPQ	24.43 ± 0.41	63.73 ± 0.27	94.20 ± 0.16	46.14 ± 0.50	62.27 ± 0.36	80.97 ± 0.27
ERVQ	22.37 ± 0.34	60.48 ± 0.43	92.58 ± 0.17	42.31 ± 0.80	57.91 ± 0.98	76.77 ± 0.96
ERVQ	23.63 ± 0.37	63.25 ± 0.46	93.92 ± 0.24	45.43 ± 0.40	61.64 ± 0.51	80.11 ± 0.43
CQ	16.29	49.95	85.59	33.73	48.03	66.25
AQ	26	70	95	–	–	–
LSQ-32	$\mathbf{29.54} \pm 0.33$	$\mathbf{72.65} \pm 0.51$	$\mathbf{97.34} \pm 0.14$	$\mathbf{55.10} \pm 0.38$	$\mathbf{72.09} \pm 0.43$	$\mathbf{88.78} \pm 0.25$

Table 3.1: Detailed recall@N values for our method on the SIFT1M dataset.

where it is immediately clear that LSQ outperforms the classical baselines, PQ and OPQ, in recall@N for all values of N. Similarly, our method outperforms CQ when using 16 ILS iterations, and the gap is widened when using 32 iterations. As we will see, analogous effects are observed throughout our results. In Table 3.1 we show our results in more detail, providing mean recall@N values and standard deviations for the SIFT1M dataset.

We also observe that CQ performs much worse than reported in the original publication, and in fact its recall is lower than the classical baselines PQ and OPQ. This is due to the difference in the size of the training set used in the publication (the entire base set), versus what is normally used in the literature ($10\times$ smaller training set). This suggests that CQ is not very sample efficient, as it needs more data than the rest of the baselines to generalize.

On Convnet1M, the advantage of LSQ over PQ, OPQ and other baselines is more pronounced. When using 64 bits, our method achieves a recall of 18.64, more than double that of PQ at 7.13, and with an 81% improvement over OPQ at 10.28. These results show an increased advantage of our method over orthogonal approaches in deep-learned features, which currently dominate computer vision applications.

3.6.2 Query/base datasets

In Figure 3.3 and Table 3.2, we report results on datasets with query/base partitions: MNIST and LabelMe22K. Our method also outperforms the state of the art on both datasets. We also note that CQ again performs 2 to 5 points below the results reported in the original publication, and in this case, this discrepancy cannot be

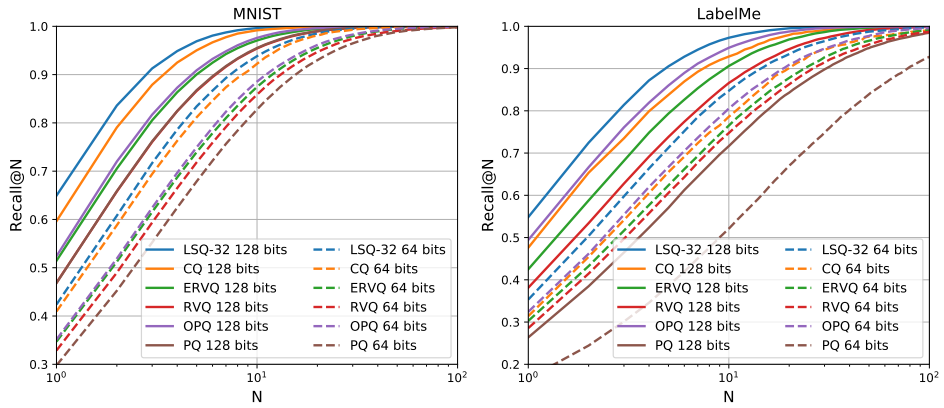

Figure 3.3: Recall@N curves for (left) the MNIST, and (right) LabelMe22K datasets.

	MNIST – 64 bits			LabelMe22K – 64 bits		
	R@1	R@2	R@5	R@1	R@2	R@5
PQ	29.76 ± 0.37	45.24 ± 0.33	67.88 ± 0.63	16.63 ± 0.46	24.43 ± 0.73	38.61 ± 0.74
OPQ	35.13 ± 0.65	51.89 ± 0.64	74.95 ± 0.47	32.43 ± 0.78	46.17 ± 0.92	66.70 ± 1.08
RVQ	32.85 ± 0.70	48.98 ± 0.82	71.83 ± 0.83	28.56 ± 0.97	41.47 ± 1.22	60.57 ± 1.68
ERVQ	34.70 ± 0.45	51.25 ± 0.62	73.95 ± 0.58	30.28 ± 0.91	43.25 ± 1.05	62.36 ± 1.45
CQ	40.94	58.80	81.16	31.40	45.45	65.50
LSQ-8	**42.38** ± 0.49	**60.78** ± 0.38	**83.42** ± 0.28	**35.32** ± 1.13	**50.39** ± 1.54	**71.25** ± 0.70

Table 3.2: Detailed recall@N on the MNIST and LabelMe22K datasets.

attributed to a difference in training protocol. Unfortunately, this means that the results reported by Zhang et al. (2014) cannot be reproduced even when using the code provided by the original authors.

3.6.3 Very large-scale training/query/base datasets

Finally, we report results on two very large-scale datasets: SIFT10M and SIFT1B in Figure 3.4, with detailed results in Table 3.3. On SIFT1B with 64 bits, LSQ-32 shows a relative improvement of 13% in recall@1 over CQ, our closest competitor, and consistently obtains between 2 and 3 points of advantage in recall over CQ in other cases. These are, to the best of our knowledge, the best results reported on SIFT1B using exhaustive search so far.

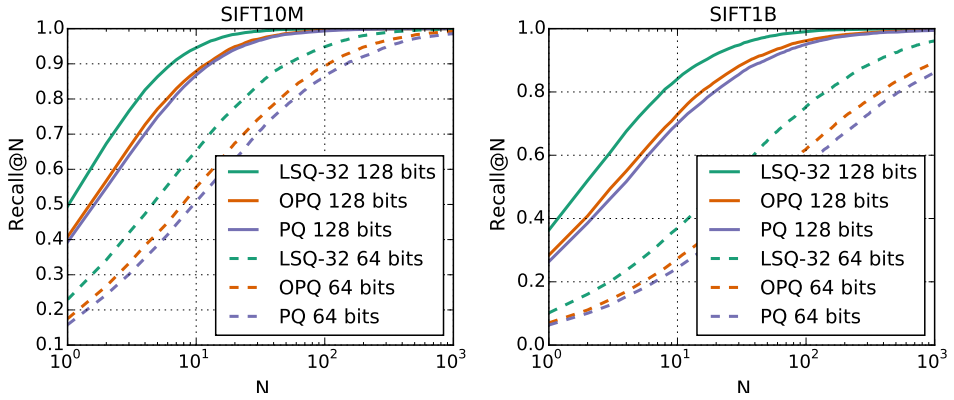

Figure 3.4: Recall@N curves for very large-scale datasets: (left) the SIFT10M, and (right) SIFT1B.

	SIFT10M – 64 bits			SIFT10M – 128 bits		
	R@1	R@10	R@100	R@1	R@2	R@5
PQ	15.79	50.86	86.57	39.31	54.74	74.89
OPQ	17.49	54.92	89.41	40.80	56.73	77.15
CQ	21	63	93	47	64	84
LSQ-16	22.51	64.62	94.67	49.26	66.75	85.36
LSQ-32	**22.94**	**65.20**	**94.85**	**49.50**	**67.31**	**86.33**

	SIFT1B – 64 bits			SIFT1B – 128 bits		
	R@1	R@10	R@100	R@1	R@2	R@5
PQ	06.34	24.41	56.92	26.38	38.57	56.45
OPQ	07.02	27.34	61.89	28.43	40.80	59.53
CQ	09	33	70	34	48	68
LSQ-16	09.73	35.82	73.84	35.32	50.84	70.66
LSQ-32	**10.18**	**36.96**	**75.31**	**36.35**	**51.99**	**72.13**

Table 3.3: Detailed recall@N values for our method on large-scale datasets: SIFT10M and SIFT1B.

3.6.4 Sparse extension

Following Zhang et al. (2015), we evaluated the sparse version of our method using 2 different levels of sparsity: SLSQ1, with $\|C\|_0 \leq S = h \cdot d$, which has a query time comparable to PQ, and SLSQ2, with $\|C\|_0 \leq S = h \cdot d + d^2$, which has a query time comparable to OPQ. We compare against SCQ1 and SCQ2 by Zhang et al. (2015), which have the same levels of sparsity. A detailed study of the relationship between sparsity and query times for very-large-scale datasets has already been presented in Zhang et al. (2015), and it is clear that our improved encoding method has no effect on query time, so we do not repeat those experiments. Instead, we focus on a smaller dataset, SIFT1M. As noted in Section 3.4, substantial speedups can also be obtained on small datasets by using sparse codebooks.

Again, in this scenario we observed improved performance compared to our

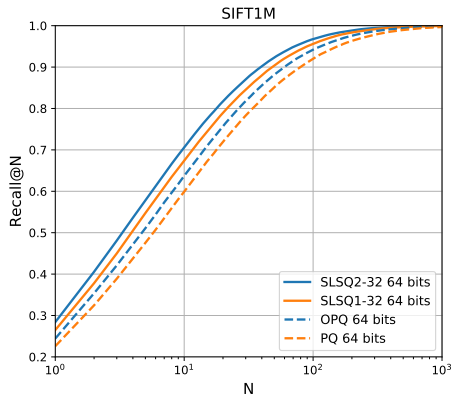

Figure 3.5: Recall@N curves for our sparse methods SLSQ1 and SLSQ2 on SIFT1M.

	SIFT1M – 64 bits		
	R@1	R@10	R@100
PQ	22.59 ± 0.43	59.90 ± 0.28	91.98
SCQ	25	67	95
SLSQ1-16	25.89 ± 0.25	66.49 ± 0.37	95.25 ± 0.18
SLSQ1-32	**26.49** ± 0.31	**67.39** ± 0.30	**95.59** ± 0.22
OPQ	24.43 ± 0.41	63.73 ± 0.27	94.20 ± 0.16
SCQ2	27	68	96
SLSQ2-16	27.87 ± 0.23	69.85 ± 0.28	96.47 ± 0.15
SLSQ2-32	**28.40** ± 0.17	**70.59** ± 0.29	**96.76** ± 0.22

Table 3.4: Recall@N values for the sparse extension of our method on SIFT1M using 64 bits.

baselines. Our improvement is of 1.49 and 1.40 over SCQ. This is close to the 1.84 gain that OPQ achieves over PQ in the same dataset.

3.6.5 Encoding speed

Method	Sequential		Batched	
codebooks (m)	7	15	7	15
LSQ-16 (ms.)	1.52	7.02	0.53	2.01
LSQ-32 (ms.)	3.01	13.93	1.05	4.03

Method	GPU (batched)	
codebooks (m)	7	15
LSQ-16 (μs)	17.9	67.2
LSQ-32 (μs)	35.7	134.3

Method	Exhaustive NN	
codebooks (m)	8	16
PQ (μs)	42.6	77.9
OPQ (μs)	49.2	90.3

Table 3.5: Time spent per vector during encoding in our approach. "Sequential" refers to an LSQ implementation where ICM encodes each point sequentially (i.e., does not take advantage of the shared pairwise terms). "Batched" is our LSQ implementation, which performs conditioning of shared pairwise terms among several data points.

In Table 3.5 we show the speed advantage that sharing the pairwise terms gives to LSQ over a naïve implementation that encodes each point sequentially. The table shows that our method, implemented in Julia (Bezanson et al., 2014) with some C++ bindings, remains fast when using up to 32 ILS iterations, handily achieving speeds faster than real-time (we believe that even better performance could be achieved with a C implementation). With our GPU implementation, it is possible

to encode SIFT1B using 128 bits and 32 ILS iterations in around 1.5 days.

3.7 Conclusion

We have introduced a new method for solving the encoding problem in Additive quantization (AQ) based on Stochastic local search (SLS). The high encoding performance of our method demonstrates that the elegant formulation introduced by AQ can be leveraged to achieve an improvement over the current state of the art in Multi-codebook quantization (MCQ). We have also shown that our method can be easily extended to accommodate sparsity constraints in the codebooks, which results in another conceptually simple method that outperforms its competitors.

Chapter 4

Stochastic relaxations and fast codebook updates

Evils coming 'round at five
To get twice as much done in half the time
—Street Chant

A wide range of optimization methods for multi-codebook quantization (MCQ) alternatively update the codebooks and codes until convergence. A common problem with such iterative methods is their tendency to stagnate in local minima; for example, in the single-codebook case, equivalent to k-means, it is known that these local minima may be arbitrarily bad (Arthur and Vassilvitskii).

In this chapter, we introduce two inexpensive approximations of simulated annealing (a well-established method for escaping local minima) that are directly applicable to MCQ. Our methods are inspired by a family of techniques used to improve k-means clustering that date back to the work of Linde et al. (1980), and were formalized by Zeger et al. (1992). This first contribution leads to improved (i.e., lower-error) MCQ solutions, with a negligible increase in computational cost.

We also focus on tackling the computational cost of training in local search qunatization (LSQ), which is a major barrier to its applicability to very large-scale datasets. To this end, we introduce a codebook update method that is 1 to 2 orders of magnitude faster than previous work. This novel method is based on two key

observations:

1. It is possible to use a direct regularized least-squares method based on Cholesky decomposition to update the codebooks (rather than the iterative conjugate gradient (CG) methods used in previous work), and

2. The computational bottleneck of this process consists of a transposition and multiplication of a very large binary matrix, and both of these operations can be accelerated without the need to explicitly build large sparse matrices.

Overall, our second contribution results in reduced training times.

Our improvements directly improve LSQ, which we introduced in Chapter 3.

4.1 Background and related work

The first of our improvements deals with the codebook update step of MCQ. The second is a technique inspired in simulated annealing which, in the past, has been used to improve k-means clustering. We review both areas next.

4.1.1 Codebook update in MCQ

Updating one codebook at a time

Greedy MCQ optimization approaches such as RVQ (Chen et al., 2010) and ERVQ/SQ (Ai et al., 2014; Martinez et al., 2014) update one codebook at a time. Updating a single codebook in MCQ is equivalent to the codebook update step in k-means, which means that this step is not particularly computationally expensive. Moreover, the best performing MCQ methods use Expectation-maximization (EM)-like optimization. In this case, the codebook update step amounts to solving a large-scale least-squares problem. In this chapter we focus on improving the codebook update step commonly used in EM-like MCQ optimization.

Updating all codebooks at once

Formally, updating all the codebooks in MCQ in an EM-like approach amounts to determining

$$\min_{C} \|X - CB\|_F^2. \tag{4.1}$$

When introducing AQ, Babenko and Lempitsky (2014a) noted that this problem can be solved independently across each dimension:

> While the least-squares optimization [...] may seem large (given large n and $[d]$), one can observe that it decomposes over each of the d dimensions. Thus, instead of solving a single large-scale least squares problem with $[mhd]$ variables, it is sufficient to solve $[d]$ least-squares problems with $[mh]$ variables, which can be formulated as an overconstrained system of linear equations. [...] As a result, the complexity of codebook learning is dominated by the encoding step.

The authors further proposed to use a large-scale conjugate-gradient solver (Fong and Saunders, 2011), and in practice built a sparse representation of B which was passed to the solver, and reused for all d problems.

Codebook update in CQ. Zhang et al. (2014) also used the AQ formulation to initialize their main method (CQ). The authors derive a closed-form solution for the problem: $C = XB^\top(BB^\top)^{-1}$, but give no further details about the exact optimization procedure. Three years later, looking at the CQ code release due to Zhang et al. (https://github.com/hellozting/CompositeQuantization), we have realized that the matrix inversion is done with the help of a singular value decomposition, ignoring solution components associated with small eigenvalues (because the solution may not be numerically stable). Moreover, the authors build a dense matrix to represent B. Thus, this method is in practice slower than using a conjugate gradient solver, and can also benefit from the contributions that we introduce here.

Advantages of reducing running time of the codebook update step in MCQ

In MCQ, updating the codebooks and updating the codes are the two main optimization steps. In the original AQ paper, Babenko and Lempitsky noted that the running time is dominated by the encoding step. However, after introducing LSQ and its GPU encoding implementation, we noticed that on the SIFT1M dataset at 64 bits:

- updating B takes around 1.2 seconds, while

- updating C takes around 5.6 seconds.

Thus, we clearly have a practical motivation to decrease the running time of the codebook update step.

4.1.2 Improving k-means

To put stochastic relaxations in context, we review them next to several other techniques that improve the standard k-means algorithm; such improvements can be broadly grouped in four categories.

1. *Encoding speed.* In k-means encoding amounts to finding, for each vector in the database, its nearest neighbour in a given codebook. For large datasets, large codebooks or high vector dimensionality, this can become a major computational bottleneck. Approaches to speed up encoding take advantage of the triangle inequality (Elkan, 2003) or convergent low-dimensional bounds on metric spaces (Hwang et al., 2012a) to reduce the number of codebook vectors to which exact distance has to be computed. In the computer vision community, Philbin et al. (2007) popularized the use of randomized kd-trees for approximate encoding to build bag-of-visual-words representations.

2. *Initialization.* A typical way to initialize the k-means algorithm is to produce an initial codebook by sampling points from the dataset uniformly at random. Multiple alternatives for initialization have been proposed (for a recent survey, see the work by Celebi et al. (2013)), and perhaps the most popular initialization algorithm is currently k-means++ (Arthur and Vassilvitskii), which also offers theoretical guarantees on the quality of the initialization.

3. *Soft assignments.* It is also possible to assign each database vector to multiple entries in a codebook using multiple, weighted assignments (Dunn, 1973; Jain, 2010). According to Jain (2010), the idea can be attributed to Dunn (1973). Martinetz et al. (1993) used a 'neural gas' network with a softmax layer for soft cluster assignments, which allowed for batched optimization using backpropagation.

4. *Stochastic relaxations (*SR*).* Several methods have exploited the idea of injecting randomness into the optimization process of k-means in order to improve the quality of the solution[1] (Flanagan et al., 1989; Linde et al., 1980; Vaisey and Gersho, 1988); we follow Zeger et al. (1992) and refer collectively to these methods as Stochastic eelaxation (SR). The idea is to exploit Simulated annealing (SA), which is guaranteed to eventually find a globally optimal solution but in practice requires extremely slow temperature decay schedules, leading to impractical (exponential) running times. Therefore, the research challenge is to find SA approximations that produce good solutions at reasonable computational costs.

The first three approaches (improving encoding speed, improving initialization or using soft assignments) are not trivially adaptable to MCQ. For example, it is not immediately clear how the triangle inequality or kd-trees could be used on the combinatorial number of centroids created by MCQ. On the other hand, initialization approaches typically add one codebook entry at a time (Arthur and Vassilvitskii); this is problematic in our case, as in MCQ, adding one entry in a codebook creates exponentially many new centroids. Finally, there is no straightforward way of listing multiple assignments on a combinatorial space in a computationally efficient way.

Fortunately, some SR methods can be formalized by adopting a functional view of k-means (Zeger et al., 1992), in which it is assumed that the optimization state is fully observable given one of the two latent variables in the problem. In Section 4.2.2, we define a functional view of MCQ, which allows us to successfully extend SR to MCQ.

[1] Notably, in their seminal work on vector quantization, Linde et al. (1980) mention an example where adding gradually decreasing amounts of Gaussian noise to the training examples results in better solutions. The authors "[...] conjecture that this technique will work more generally [...] but [...] have been unable to prove this theoretically". See (Linde et al., 1980, p. 93), for more details.

4.2 Solution methodology

Our first contribution deals with the codebook update step, which amounts to determining

$$\min_{C} \|X - CB\|_F^2. \tag{4.2}$$

The current state-of-the-art method for this step was originally proposed by Babenko and Lempitsky (2014a), who noticed that finding C corresponds to a least-squares problem where C can be found independently in each dimension. The authors further use an iterative conjugate gradient method (Fong and Saunders, 2011) in this step, which has the additional advantage that B can be reused for the d problems that finding C decomposes into. We have identified two problems with this approach:

1. *Explicit sparse matrix construction is inefficient.* Using conjugate gradient methods requires that B be converted into an explicit sparse matrix. Although efficient data structures for sparse matrices exist (e.g., the compressed sparse row of numpy), in practice, B is stored as an $m \times n$ `uint8` matrix. Ideally, we would like to use B directly and avoid using an additional data structure.

2. *Failure to exploit the binary nature of B.* Matrix B is composed exclusively of ones and zeros (i.e., it is binary). Data structures used for sparse matrices are commonly designed for the general case when the non-zero entries are arbitrary real numbers, which leaves room for additional optimization.

4.2.1 Direct codebook update

We now introduce our method for fast codebook update, which takes advantage of these two observations. First, we note that it is possible to use a numerically-stable direct method by rewriting Expression 4.2 as a regularized least-squares problem:

$$\min_{C} \|X - CB\|_F^2 + \lambda \|C\|_F^2, \tag{4.3}$$

In this case, the optimal solution can be obtained by taking the derivative with respect to C and setting it to zero, as is commonly done in ridge regression

$$C = XB^\top (BB^\top + \lambda I)^{-1}. \tag{4.4}$$

While we are not interested in a regularized solution, we can still benefit from this formulation by setting λ to a very small value ($\lambda = 10^{-4}$ in our experiments), which simply renders the solution numerically stable.

We further note that the matrix $BB^\top + \lambda I \in \mathbb{R}^{mh \times mh}$ is square, symmetric, positive-definite and fairly compact; *notably, its size is independent of n*. Thus, matrix inversion can be performed directly via Cholesky decomposition in $O(m^3h^3)$ time. Since matrix inversion is efficient, the bottleneck of our method lies in computing $BB^\top \in \mathbb{N}^{mh \times mh}$, as well as $XB^\top \in \mathbb{R}^{d \times mh}$. We exploit the structure in B to accelerate both operations.

Computing $BB^\top$. By indexing B across each codebook, $B = [B_1, \cdots, B_m]^\top$, $BB^\top$ can be written as a block-symmetric matrix composed of m^2 blocks of size $h \times h$ each:

$$BB^\top = \begin{bmatrix} B_1B_1^\top & B_1B_2^\top & \dots & B_1B_m^\top \\ B_2B_1^\top & B_2B_2^\top & \dots & B_2B_m^\top \\ \vdots & \vdots & \ddots & \vdots \\ B_mB_1^\top & B_mB_2^\top & \dots & B_mB_m^\top \end{bmatrix}, \tag{4.5}$$

here, the diagonal blocks $B_NB_N^\top$ are diagonal matrices themselves, and since B is binary, their entries are a histogram of the codes in B_N. Moreover, the off-diagonal blocks are the transpose of their symmetric counterparts: $B_NB_M^\top = (B_MB_N^\top)^\top$, and can be computed as bivariate histograms of the codes in B_M and B_N. Using these two observations, this method takes $O(m^2n)$ time, while computing $BB^\top$ naïvely would take $O(m^2h^2n)$.

Computing $XB^\top$. We again take advantage of the structure of B to accelerate this step. $XB^\top$ can be written as a matrix of m blocks of size $d \times h$ each,

$$XB^\top = [XB_1^\top, XB_2^\top, \ldots, XB_m^\top]. \tag{4.6}$$

Each block $XB_i^\top$ can be computed by treating the $B_i^\top$ columns as binary vectors that select the columns of X to sum. This method takes $O(mnd)$ time, while computing $XB^\top$ naively would take $O(mhnd)$.

4.2.2 Stochastic relaxations (SR)

Broadly defined, simulated annealing (SA) is a classical stochastic local search (SLS) technique (Hoos and Stützle, 2004) that requires 4 major components:

1. **Intitialization**: A function that creates the initial candidate solution s.
2. **Neighbourhood**: The space of candidate solutions s' that may be explored from a starting solution s.
3. **Temperature schedule**: A function that controls the probability to accept or reject a new (worse) solution. (Better solutions are typically always accepted). The temperature is normally decreased as the optimization progresses. Various temperature schedules have been proposed and used in the many applications of simulated annealing.
4. **Termination criterion**: A condition that must be satisfied for the search to end.

A stochastic relaxation (Zeger et al., 1992) relaxes some of the typical SA steps in order to make them more computationally efficient. For component (1), we initialize our solutions uniformally at random, and for component (4) we simply limit the number of search iterations to a constant.

We now formally define our optimization state s, perturbation procedure $\pi(\cdot)$ and acceptance criterion, for our method, which form the remaining components of our SA approximation.

Defining a SA state: a functional view of MCQ

We formally define an optimization state in MCQ. Since Expression 2.4 is defined over two latent variables, C and B, a naïve approach is to define the state at time

step i as a 2-tuple $s_i = (C_i, B_i)$. However, it is not clear how one could use this definition to perturb the state: for example, adding random noise to B would alter its discrete nature, taking it out of the space of feasible solutions.

Moreover, SLS methods are often designed to balance two main factors: (a) the amount of perturbation that allows them to escape from local minima, and (b) the ability of a local search method to recover from such perturbations and, potentially, find new and improved solutions. Globally, EM-like MCQ methods already rely on powerful local search techniques, namely the codebook-update and encoding methods. This suggests that these functions can be used as a subroutines in a global method that improves the quality of the solution. Following this idea, the first step in SR is to make the assumption that the optimization state is fully observable given a single variable, either C or B, which fully specifies the other via a function. We thus define two functions:

- an "encoder" function $\mathscr{C}(X, B) \to C$, and
- a "decoder" function $\mathscr{D}(X, C) \to B$.

Notice that, in k-means, $\mathscr{C}$ updates the centroids by taking the means of its assignments, and $\mathscr{D}$ assigns each training point to its nearest neighbour in the codebook. In MCQ, $\mathscr{C}$ amounts to the codebook-update step, for which we adopt the fast ridge regression method described in Section 4.2.1. Similarly, $\mathscr{D}$ amounts to updating the codes B; in this case, we adopt the encoding method introduced in Chapter 3, based on Iterated local search (ILS) with ICM and random perturbations.

We can now formally define the optimization state of MCQ in two ways, which will eventually give rise to two different SR methods. The first method, called SR-C, uses the encoder function $\mathscr{C}$, and the second method, called SR-D, uses the decoder function $\mathscr{D}$.

Perturbing the SA state

The next step is to define a way to perturb the SA state at time-step i. We similarly define 2 perturbation methods, one for SR-C and one for SR-D. Since we have defined the state as fully-observable given either variable via a proxy function, we can perturb the state by simply perturbing the function used in SR-C or SR-D. Formally, we define the functions

- $\mathscr{C}^* := \mathscr{C}(\pi_C(X,i),B) \to C$ for SR-C, and
- $\mathscr{D}^* := \mathscr{D}(X,\pi_D(C,i)) \to B$ for SR-D,

where the function $\pi_C(X,i) \to X + T(i) \cdot \varepsilon$ amounts to adding noise ε to X, according to a predefined temperature schedule T. In our case, we have chosen our noise to be sampled from a zero-mean Gaussian with a diagonal covariance proportional to X; in other words, $\varepsilon \sim N(\mathbf{0},\Sigma)$, where $\Sigma = diag(cov(X))$.

The main difference between k-means and MCQ is that, in MCQ, we use multiple codebooks. This difference is particularly important in SR-D, where the noise affects C, which represents m different codebooks. Since the centroids are obtained by summing one entry from each codebook, perturbing C amounts to perturbing the centroids m times. We found that this perturbation is too extreme and results in poorer solutions. Thus, in SR-D we instead use the average covariance of each codebook – this amounts to multiplying the noise by a factor of $1/m$ in SR-D. We thus define the perturbation function for SR-D slightly differently: $\pi_D(C,i) \to C + (T(i)/m) \cdot \varepsilon$. In other words, we multiply the noise by factor of $1/m$ in SR-D.

Notice that, in these functional formulations, the discrete variable B is never perturbed. Thus, our approximations remain anytime algorithms and do not require additional rounding steps.

Temperature schedule. In SA, it is common to gradually reduce the amount of noise that controls the probability of accepting of a new state (the so-called Metropolis-Hastings criterion). In SR, following Zeger et al. (1992), we instead use the temperature to control the amount of noise added in each time-step and consider three temperature schedules:

$$T(i) \to (1-(i/I))^p, \tag{4.7}$$

$$T(i) \to 1/(i+1)^p, \text{ and} \tag{4.8}$$

$$T(i) \to p^i, \tag{4.9}$$

where I is the total number of iterations, i represents the current iteration, and $p \in (0,1]$ is a tunable hyper-parameter. We experimented with values of $p =$

$\{0.1, 0.5, 1\}$ and found that the first temperature schedule (Eq. 4.7) and a value of $p = 0.5$ produce the best results. We use these parameters in all our experiments.

Acceptance criterion

The final building block of SA is an acceptance criterion, which decides whether the new (perturbed) state will be accepted or rejected. Following Zeger et al. (1992), we decide to always accept the new state. As we will show, this criterion gives excellent results in practice, and makes it very easy to control for running time, as each iteration is practically identical to LSQ.

4.2.3 Recap: SR-C and SR-D

To summarize, we have now introduced two algorithms that define crude (but, as we will see, very effective) approximations to simulated annealing: SR-C and SR-D. These approximations are also very simple to implement, and are in fact not very different in practice from LSQ. To highlight this simplicity, we have outlined the LSQ optimization pipeline in Algorithm 3:

Algorithm 3 Block coordinate descent approaches to MCQ

```
1: function LSQ(X, I)
2:     B ← initialization()
3:     i ← 1                                   ▷ Iteration counter
4:     while i < I do
5:         C ← argmin_C ||X − CB||_F^2          ▷ Codebook update
6:         B ← argmin_B ||X − CB||_F^2          ▷ Encoding step
7:         i ← i + 1
8:     end while
9:     return C, B
10: end function
```

Notice that

- **SR-C** follows Algorithm 3 exactly, except that line 5 is replaced by $C \leftarrow \text{argmin}_C \|\pi_C(X, i) - CB\|_F^2$;
- **SR-D** follows Algorithm 3 exactly, except that line 6 is replaced by $B \leftarrow \text{argmin}_B \|X - \pi_D(C, i)B\|_F^2$.

In other words, SR-C and SR-D amount to adding noise in different parts of the typical MCQ optimization pipeline, but the workhorse functions that perform the codebook-update and the encoding encoding step remain unchanged. As a result, if in the future better codebook-update or encoding functions are developed, they can be seamlessly integrated into our pipelines. Finally, we note that our methods involve only minimal computational overhead, as they only require the computation of the covariance of either X (which can be computed once and re-used many times in (SR-C)), or C, which is a compact variable independent of n. In practice, this overhead is negligible: < 0.1 seconds for SR-C, and < 0.01 seconds for SR-D.

4.3 Experiments

We now describe the evaluation protocol that we use to empirically validate our contributions.

Datasets and protocols

We evaluate our approach on five datasets. The first three are SIFT1M (Jégou et al., 2011), LabelMe22K (Russell et al., 2008) and MNIST (LeCun et al., 1998). These correspond to handcrafted features (SIFT and GIST), and small raw images (MNIST). The next two datasets are based on features extracted from deep convolutional neural networks. The VGG dataset is extracted from the last layer of the CNN-M-128 network provided by Chatfield et al. (2014); this network internally compresses the features by limiting the dimensionality of the output to 128 units. Finally, we put together the Deep1M dataset, by sampling from the 10 million example set provided with the recently introduced Deep1B dataset (Babenko and Lempitsky, 2016). These features come from the last convolutional layer of a network similar to GoogLeNet v3 (Szegedy et al., 2014), and have been PCA-projected to 96 dimensions. Table 4.1 summarizes the datasets used in our ANN experiments.

We follow the same protocol as in the previous chapter, and use the training set to learn the codebooks C; we then use those codebooks to encode the base set (i.e., obtain B), and finally use the query set to find approximate nearest neighbours in

Dataset	d	# training	# query	# base
SIFT1M	128	100 000	10 000	1 000 000
LabelMe22K	512	–	2 000	20 019
MNIST	784	–	10 000	60 000
Deep1M	96	100 000	10 000	1 000 000
VGG	128	100 000	10 000	1 000 000

Table 4.1: Statistics of the datasets used in our ANN experiments.

the compressed base set. MNIST and LabelMe22K, however, do not have a train set, so we use the base set to learn both C and B. This protocol is in line with previous work (Gong and Lazebnik, 2011; Zhang et al., 2014).

Evaluation

To evaluate our first contribution (i.e., fast codebook update), we measure codebook update times. We also use our improved codebook update method while reproducing the results of LSQ (introduced in Chapter 3). Obtaining the same recall@N values as in previous work while using our fast codebook update method validates that our contribution does not affect the quality of the solution.

To evaluate SR, we report recall@N on all our datasets. Recall@N represents the empirical probability over the query set that the actual nearest neighbour of the query is found within the first N retrieved entries. In the retrieval literature, recall@1 is often considered the most important value in this curve. We also run all the methods ten times on each dataset and report the mean of the five trials to control for the inherent randomness in recall@N.

Baselines

Our baselines are the orthogonal MCQ methods PQ (Jégou et al., 2011) and OPQ (Ge et al., 2014; Norouzi and Fleet, 2013), as well as the non-orthogonal RVQ (Chen et al., 2010), ERVQ/SQ (Ai et al., 2014; Martinez et al., 2014) and CQ (Zhang et al., 2014), as well as LSQ, i.e., the method that we introduced in Chapter 3. We control for memory usage by making our baselines and our method produce final codes of

64 and 128 bits. This means that PQ, OPQ and CQ use 8 and 16 codebooks, while the rest of the methods use 7 and 15 codebooks, plus an extra codebook to encode the norm of the approximation. This has the side effect of controlling for query time as well, because all the methods use 8 (resp. 16) table lookups to compute approximate distances.

4.4 Results

In this section we present the results of our experiments.

4.4.1 Fast codebook update

	64 bits		128 bits	
SIFT1M	CG	Chol	CG	Chol
Julia, C++	14.2s	5.6s	38.5s	22.5s
Julia, CUDA	6.8s	1.2s	20.3s	4.3s

	64 bits		128 bits	
Deep1M	CG	Chol	CG	Chol
Julia, C++	9.5s	7.2s	30.7s	24.2s
Julia, CUDA	3.3s	1.0s	10.6s	4.1s

Table 4.2: Total time per LSQ/LSQ++ iteration, depending on how we update C (CG or Cholesky), and how we update B (using a C++ or a CUDA implementation).

We show the total elapsed time for training once we incorporate our codebook update method on Table 4.2. Our method saves anywhere from 2.3 (Deep1M, 64 bits) to 16 seconds (SIFT1M, 128 bits) of training time per iteration. Alternatively, when encoding is GPU-accelerated, it results in $2.2-5.6\times$ speedups in practice.

Large-scale experiments

On Figure 4.1, we show a "stress-test" comparison between our method for fast codebook update and Conjugate gradient (CG), using dataset sizes of $n = \{10^4, 10^5, 10^6, 10^7\}$. We take the first n training vectors from the SIFT1B (Jégou et al., 2011) and Deep1B (Babenko and Lempitsky, 2016) datasets, and generate a random B. This is a specially easy case for CG, and it takes only 2-3 iterations to converge. Even in this case, our method is orders of magnitude faster than previous work, and stays under 10 seconds in all cases, while CQ takes up to 700 seconds for $n = 10^7$. Our

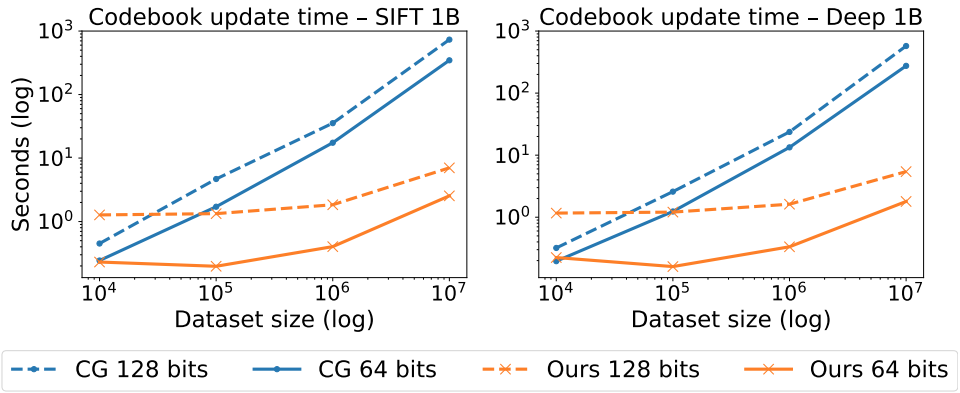

Figure 4.1: Time for codebook update as a function of dataset size with up to ten million vectors.

method is only slower on small training sets, perhaps due to the complexity of matrix inversion, which is independent of n.

4.4.2 SR-C and SR-D

SR requires the setting of several hyperparameters. First, we have to choose the perturbation method to use (either SR-C, or SR-D), and second, we have to choose the temperature schedule with its corresponding decay parameter. We report results using both methods, Eq. 4.7 for temperature decay, and $p = 0.5$. All results reported in this work have been obtained using the same hyperparameters, which makes our two contributions usable as out-of-the-box improvements on top of LSQ without hyperparameter tuning.

Experimentally, we have found that SR-D (which perturbs the codebooks) performs better on handcrafted features and on MNIST, and SR-C performs better on deep features, which may be used as simple rule of thumb for practitioners. However, SR-C also tends to be more unstable.

Query/base datasets

In Figure 4.2, we show results on the small query/base datasets MNIST and LabelMe22K, after training for 100 iterations. We also present results in detail in Table 4.3.

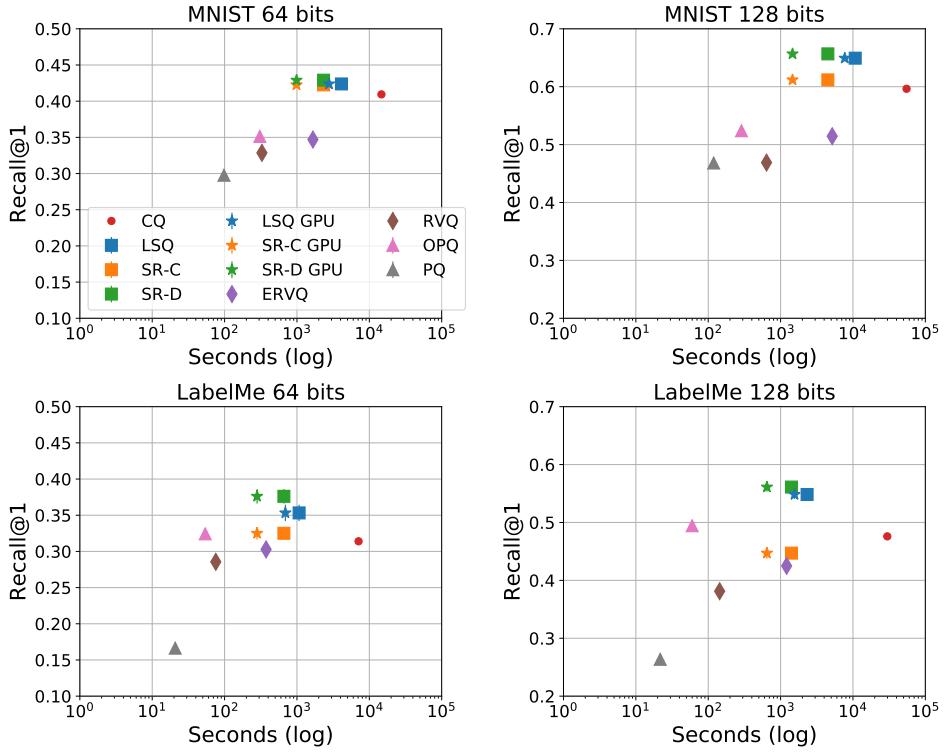

Figure 4.2: Running time vs recall@1 on the MNIST and LabelMe datasets.

	MNIST – 64 bits			LabelMe22K – 64 bits		
	R@1	R@2	R@5	R@1	R@2	R@5
PQ	29.76 ± 0.37	45.24 ± 0.33	67.88 ± 0.63	16.63 ± 0.46	24.43 ± 0.73	38.61 ± 0.74
OPQ	35.13 ± 0.65	51.89 ± 0.64	74.95 ± 0.47	32.43 ± 0.78	46.17 ± 0.92	66.70 ± 1.08
RVQ	32.85 ± 0.70	48.98 ± 0.82	71.83 ± 0.83	28.56 ± 0.97	41.47 ± 1.22	60.57 ± 1.68
ERVQ	34.70 ± 0.45	51.25 ± 0.62	73.95 ± 0.58	30.28 ± 0.91	43.25 ± 1.05	62.36 ± 1.45
CQ	40.94	58.80	81.16	31.40	45.45	65.50
LSQ-8	42.38 ± 0.49	60.78 ± 0.38	**83.42** ± 0.28	35.32 ± 1.13	50.39 ± 1.54	71.25 ± 0.70
SR-D-8	**42.90** ± 0.34	**61.25** ± 0.29	83.29 ± 0.26	**37.61** ± 1.00	**52.54** ± 0.94	**73.45** ± 0.91
SR-C-8	42.26 ± 0.26	60.79 ± 0.41	82.75 ± 0.29	32.51 ± 0.86	46.34 ± 0.78	66.33 ± 1.03

Table 4.3: Detailed recall@N on the MNIST and LabelMe22K datasets.

We can observe that SR-D always outperforms LSQ, and using the fast codebook update means that the method is also more efficient in practice. We also observe that CQ (Zhang et al., 2014) is an outlier in terms of computation – the most expensive method by far – , but is not competitive in terms of recall. Finally we also observe that, in these two datasets, SR-C often yields lower recall than LSQ.

SR-D method achieves the best improvement in LabelMe22K at 64 bits, where with 100 iterations, it obtains 0.023 (absolute) better recall@1, or a relative 6.4% improvement over the state of the art. In other cases, the gain ranges from 0.005 to 0.015 in recall@1 over the state of the art, requiring less computation than previous work.

Train/query/base datasets

In Figure 4.3, we show the recall@1 achieved by our methods on train/query/base datasets as a function of the total time they used for learning. In this Figure, we ran training for 100 iterations and base encoding for 128 iterations.

We observe a pattern similar to the previous experiment: SR-D obtains higher recall than LSQ in all cases. However, we also observe that SR-C largely outperforms both LSQ and SR-D on the VGG dataset, which has internally been compressed by the network down to 128 dimensions. This suggests that SR-C might be better suited for applications that use deep features, which currently dominate computer vision applications.

	SIFT1M – 64 bits			SIFT1M – 128 bits		
	R@1	R@10	R@100	R@1	R@2	R@5
PQ	22.59 ± 0.43	59.90 ± 0.28	91.98 ± 0.17	44.53 ± 0.40	60.49 ± 0.40	78.94 ± 0.28
OPQ	24.43 ± 0.41	63.73 ± 0.27	94.20 ± 0.16	46.14 ± 0.50	62.27 ± 0.36	80.97 ± 0.27
ERVQ	22.37 ± 0.34	60.48 ± 0.43	92.58 ± 0.17	42.31 ± 0.80	57.91 ± 0.98	76.77 ± 0.96
ERVQ	23.63 ± 0.37	63.25 ± 0.46	93.92 ± 0.24	45.43 ± 0.40	61.64 ± 0.51	80.11 ± 0.43
CQ	16.29	49.95	85.59	33.73	48.03	66.25
AQ	26	70	95	–	–	–
LSQ-128	29.85 ± 0.23	73.10 ± 0.58	97.46 ± 0.13	55.62 ± 0.38	72.85 ± 0.19	89.46 ± 0.21
SRD-128	**30.72** ± 0.29	**74.25** ± 0.36	**97.65** ± 0.10	**56.09** ± 0.35	**73.02** ± 0.27	**89.66** ± 0.33
SRC-128	30.60 ± 0.32	73.80 ± 0.34	97.50 ± 0.13	52.16 ± 0.37	69.07 ± 0.37	86.40 ± 0.35

Table 4.4: Detailed recall@N values on the SIFT1M dataset.

Finally, Tables 4.4, 4.5 and 4.6 show the detailed performance of our methods

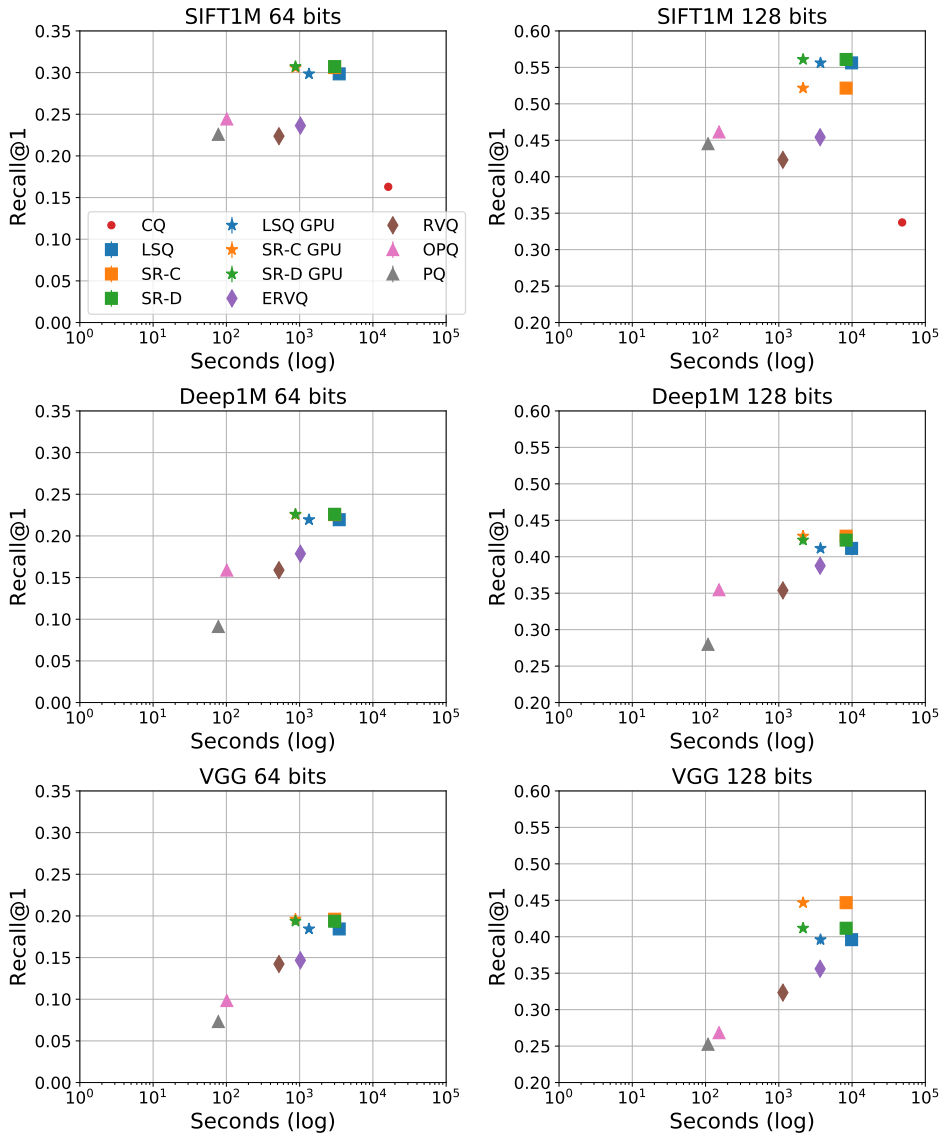

Figure 4.3: Running time vs recall@1 on the SIFT1M, Deep1M and VGG datasets.

	Deep1M – 64 bits			Deep1M – 128 bits		
	R@1	R@10	R@100	R@1	R@2	R@5
PQ	09.13 ± 0.41	33.54 ± 0.24	73.14 ± 0.54	27.98 ± 0.28	41.16 ± 0.45	60.10 ± 0.47
OPQ	15.90 ± 0.43	52.82 ± 0.47	89.61 ± 0.29	35.47 ± 0.59	50.75 ± 0.57	71.15 ± 0.31
RVQ	15.90 ± 0.53	52.64 ± 0.49	89.80 ± 0.38	35.38 ± 0.73	50.70 ± 0.91	71.45 ± 0.68
ERVQ	17.87 ± 0.37	56.73 ± 0.39	91.95 ± 0.17	38.76 ± 0.48	55.05 ± 0.54	75.76 ± 0.61
LSQ-128	21.95 ± 0.42	64.83 ± 0.61	95.30 ± 0.25	41.15 ± 0.44	57.90 ± 0.53	78.18 ± 0.37
SRD-128	**22.60** ± 0.55	**66.64** ± 0.49	**95.75** ± 0.15	42.28 ± 0.62	58.95 ± 0.57	79.23 ± 0.45
SRC-128	22.56 ± 0.33	66.30 ± 0.36	**95.75** ± 0.16	**42.83** ± 0.29	**59.77** ± 0.42	**80.01** ± 0.30

Table 4.5: Detailed recall@N values on the Deep1M dataset.

	VGG – 64 bits			VGG – 128 bits		
	R@1	R@10	R@100	R@1	R@2	R@5
PQ	07.32 ± 0.30	29.91 ± 0.45	73.39 ± 0.44	25.25 ± 0.44	37.82 ± 0.55	57.12 ± 0.32
OPQ	09.84 ± 0.18	37.68 ± 0.51	81.26 ± 0.39	26.84 ± 0.22	39.48 ± 0.25	58.89 ± 0.36
RVQ	14.23 ± 0.20	51.62 ± 0.53	93.02 ± 0.15	32.34 ± 0.81	47.72 ± 0.82	69.19 ± 0.90
ERVQ	14.67 ± 0.42	52.67 ± 0.38	93.89 ± 0.25	35.60 ± 0.31	51.36 ± 0.59	72.98 ± 0.40
LSQ-128	18.44 ± 0.26	61.13 ± 0.39	96.49 ± 0.16	39.60 ± 0.64	56.12 ± 0.52	77.38 ± 0.37
SRD-128	19.36 ± 0.43	63.59 ± 0.41	**97.48** ± 0.14	41.17 ± 0.31	58.15 ± 0.57	79.11 ± 0.22
SRC-128	**19.60** ± 0.29	**63.87** ± 0.32	97.43 ± 0.13	**44.67** ± 0.45	**62.62** ± 0.37	**83.60** ± 0.23

Table 4.6: Detailed recall@N values on the VGG dataset.

and a comparison with previous work. We can again observe that the SR variants consistently obtain state-of-the-art performance when the computational budget is identical to previous work. SR-C works specially well on the VGG dataset; for example, when using 64 bits, SR-C gains 0.016 (relative 6.2%) in Recall@1 and 0.05 points (relative 12.8%) when using 128 bits with respect to LSQ. In this challenging dataset, OPQ obtains a much smaller improvement of 0.015 (resp 0.016) over PQ. Finally, we observe a modest improvement of SR on SIFT1M, consistent with previous work (Babenko and Lempitsky, 2016) which has shown that hand-crafted features show small improvements with non-orthogonal codebooks.

4.4.3 Comparison against Competitive quantization (COMPQ)

Competitive quantization (COMPQ) (Ozan et al., 2016a) is an MCQ method based on Stochatic gradient descent (SGD) with a batch size of one. To bypass the non-diffenrentiability of the encoding step, the authors consider the chosen codes as the

single sample of a function that produces soft assignments into the codebooks. This approximation can be seen as a single-sample estimate of a policy gradient (Sutton et al., 2000).

So far, in our evaluation we have omitted a direct comparison with COMPQ. This might strike the reader as odd, since COMPQ reports the best result to date on SIFT1M (recall@1 of 0.352). Part of the reason is that the experiments reported by Ozan et al. (2016a) consider a quadratic query time, while in our experiments we have set one codebook aside to keep query time linear – and thus comparable with our baselines.

Since there is no publicly available implementation of COMPQ, we have tried to reproduce the reported results ourselves, with moderate success. We have not, for example, been able to reproduce the transform coding initialization reported in the paper, but have instead used RVQ (Chen et al., 2010), which Ozan et al. reported to achieve slightly worse results. We obtained a recall@1 of 0.346 using 250 epochs (the parameter used in the best reported result). The small difference in recall may be attributed to our different initialization.

However, the largest barrier to experimentation on our side is that our COMPQ implementation, written in Julia, takes roughly 40 minutes per epoch to run. This means that our experiment on SIFT1M with 64 bits took almost one week to finish. We contacted the COMPQ authors, and they mentioned using a multithreaded C++ implementation with pinned memory, an ad-hoc sort implementation, and special handling of threads. Their implementation takes 551 seconds per epoch, or about 38 hours ($\sim$ 1.5 days) in total for 250 epochs on a desktop with a 10-core Xeon E5 2650 v3 @2.3 GHz CPU. This is over three orders of magnitude more computationally expensive than Product quantization (PQ) (Jégou et al., 2011), a baseline considered in the same publication.

We compare COMPQ to LSQ and SR-D using a multithreaded C++ encoding implementation, as well as our GPU implementation, and show our results on Table 4.7. We use $m = 8$ codebooks in total, which controls for query time and memory use with respect to COMPQ. With only 25 training iterations and 32 ILS iterations, SR-D narrows the difference between COMPQ and LSQ by 50%, from 0.012 to 0.006, and is also overall faster due to the faster codebook update.

We iteratively double the computational budget of SR-D (trading off compu-

	Iters	Init	Training	Base encoding	Total	R@1
COMPQ (C++)	250	–	–	–	38 h	0.352
LSQ (Julia, C++)	25	2.6 m	6.34 m	5.8 m (32 iters)	15.2 m	0.340
LSQ (Julia, CUDA)	25	1.1 m	2.8 m	29 s (32 iters)	4.4 m	0.340
SR-D (Julia, CUDA)	25	1.1 m	33 s	29 s (32 iters)	2.1 m	0.346
SR-D (Julia, CUDA)	50	2.2 m	1.1 m	58 s (64 iters)	4.3 m	0.348
SR-D (Julia, CUDA)	100	4.4 m	2.2 m	1.9 m (128 iters)	8.5 m	0.351
SR-D (Julia, CUDA)	100	4.4 m	2.2 m	3.9 m (256 iters)	10.5 m	0.353

Table 4.7: Comparison between CompQ, LSQ and LSQ++ on SIFT1M using 64 bits.

tation for accuracy), and bring the difference in recall to 0.001 with 100 training iterations and 128 Base encoding ILS iterations. Doubling the budget of this final step puts our method above CompQ by 0.001 in recall@1. Even under these circumstances, SR-D is 200× faster than COMPQ. Finally, we note that, since COMPQ is based on SGD with a batch size of one, it is unlikely to benefit from the parallel architectures found in modern GPUs, so the gap in speed with our method is not easy to close.

4.5 Conclusions and future work

We have introduced two stochastic relaxation methods for MCQ that provide inexpensive approximations to simulated annealing. The methods consistently improve upon the state of the art, without requiring extra computation. We have also introduced a method for fast codebook update that results in faster training in MCQ. Both our improvements can be used as out-of-the-box improvements on top of LSQ, and define a new state of the art in Multi-codebook quantization (MCQ).

It is intuitively appealing to think of Stochastic eelaxation (SR) as an approximation to Simulated annealing, and it would be interesting to further investigate this theoretically. In particular, we have noticed that the noise introduced in SR results in a perturbation of the parameters that define the encoding Markov Random Fields, which suggests a connection between our method and the perturb-and-MAP framework of Papandreou and Yuille (2011). Exploring this theoretical connection

is an interesting area of future work.

Finally, we note that while the bottleneck of our method now lies in updating the codes, for Composite quantization (CQ) (Zhang et al., 2014), the computational burden is in the codebook update step. Future improvements to CQ may focus on using recent efficient optimizers for large-scale continuous problems, such as the stochastic average gradient (Schmidt et al., 2017).

Chapter 5

Automatic hyperparameter optimization of MCQ algorithms

I think my spaceship knows which way to go
—David Bowie

The most computationally challenging part of multi-codebook quantization (MCQ) optimization is the encoding step, which amounts to maximum-a-posteriori (MAP) estimation of multiple fully-connected Markov random fields (MRFs). In Chapter 3, we introduced a method based on iterated local search (ILS) for this problem. We have also introduced a graphics processor unit (GPU) implementation of our encoding algorithm, which reduces the encoding time in practice.

In Chapter 4, we also introduced a method for codebook update which is orders of magnitude more efficient than conjugate gradient methods. Together with our GPU encoding implementation, these contributions greatly reduce training time in MCQ. We further introduced a simple modification to our algorithm that adds noise to either the data or the codebooks, according to a pre-defined schedule, resulting in improved results.

The algorithms introduced in Chapters 3 and 4 have a number of hyperparameters that control their behaviour and affect the quality of the optimization. Examples of these hyperparameters include the number of iterations in ILS, the number of iterated conditional modes (ICM) iterations, and the formula of the temperature

decay, which is itself parameterized by a scalar that controls the decay speed. So far, we have set all the hyperparameters to the same values in all our experiments, tunning their values from exhaustive search in a small search range. Thus, the performance attained by our algorithms demonstrates their robustness across a range of different input distributions including raw images, handcrafted image features and features learned by deep convolutional neural networks. This robustness is important to make our algorithms available to a large user base, and gives empirical evidence on the performance of the algorithm as out-of-the-box tools for large-scale MCQ.

However, one of our main use cases of interest is the preprocessing of very large-scale datasets (with $n \geq 10^9$ vectors) for fast approximate nearest neighbour search (ANNS). In this case, it is reasonable for a practitioner to spend time finding a set of hyperparameters that work best on the distribution of interest. This dataset-specific hyperparameter tuning is already commonplace in algorithms that perform short-listing on very large datasets such as the Inverted File (Jégou et al., 2011) and the inverted multi-index (Babenko and Lempitsky, 2012). Importantly, Matsui et al. (2015) found that the hyperparameters of these large-scale indices have a large influence in their performance, which motivated them to develop the PQTable (Matsui et al., 2015), a hyperparameter-free data structure for large-scale indexing.

Similarly, previous MCQ algorithms such as composite quantization (CQ) and sparse composite quantization (SCQ) (Zhang et al., 2014, 2015), require the setting of several hyperparameters, and we have learned, after inspecting the experimental source code, that the performance reported in their respective publications was attained after setting these hyperparameters to different values for each dataset – a detail that is unfortunately missing in the publications. Therefore, performing dataset-specific hyperparameter optimization of our algorithms also amounts to benchmarking on equal grounds with previous work.

In this chapter, we explore the use of meta-algorithms for automatic hyperparameter tuning, and apply them to MCQ. Our goal is to discover the performance potential of our algorithms when they are adapted to specific input domains.

5.1 Background and related work

Machine learning algorithms typically have both parameters and hyperparameters. The parameters of the model are the set of values that are found by an optimization algorithm, while the hyperparameters are values that control either the space and size of the model, or the behaviour of said optimization algorithm.

For example, in a deep neural network, the parameters include the weights of each layer, such as the convolutional filters or linear transformations, which are typically learned via Stochatic gradient descent (SGD). The hyperparameters include the architecture of the network itself, the learning rate of SGD and the method for weight initialization.

Automatic hyperparameter optimization

Since hyperparameters are not typically optimized by an automated algorithm, it is common for researchers to set the hyperparameters of an algorithm "by hand"; that is, by trial and error on a small set of values, or by grid search (exhaustive combinatorial enumeration over a predefined range). Prominent deep learning techniques (Krizhevsky et al., 2012; Sutskever et al., 2014) owe their success, at least partially, to the finding of a set of hyperparameters that are stable enough such that manual hyperparameter tuning does not constitute a barrier to achieve state-of-the-art results on a number of benchmarks.

In contrast, automatic hyperparameter optimization techniques (Bergstra et al., 2011; Hutter et al., 2007, 2011; Snoek et al., 2012) aim to replace the hand tuning of hyperparameters with more principled algorithms that, given a fixed computational budget, trade-off exploitation and exploration to find the best hyperpameters for the task at hand.

Most methods for hyperparameter optimization are based on black-box optimization; i.e., they have no access to the details of the function (or algorithm) that they are trying to optimize, other than the function output for a given set of (hyper) parameters. Some methods thus internally model the optimization landscape using different underlying assumptions, or use simple heuristics to probe the hyperparameter space.

ParamILS

Hutter et al. (2007) treat the hyperpameters of an algorithm as parameters to optimize in an iterated local search (ILS) framework. Given an initial hyperparameter configuration, ParamILS iteratively changes a single parameter (i.e., considers the one-exchange neighbourhood), and uses iterative first improvement as a surrogate local search procedure. The new solution (set of hyperparameters) is accepted if it yields better or equally good results. After each iteration, the search is randomly restarted from scratch with a small probability. Hutter et al. demonstrate good results optimizing Boolean satisfiability (SAT) solvers and randomly generated binary Bayesian networks.

SMAC

Hutter et al. (2011) introduce Sequential model-based optimization for algorithm configuration (SMAC), which uses a random forest to predict expected hyperparameter performance. Sequentail model-based optimization, often known as "Bayesian optimization" in the machine learning literature (Brochu et al., 2010), often uses Gaussian processes to model the optimization landscape. Using random forests instead of Gaussian processes allows a more elegant approach to deal with integer and categorical parameters, and provides a natural way to quantify uncertainty in the estimation. The authors demonstrate performance on par with or superior to ParamILS on multi-instance algorithm configuration as applied to distributions of Boolean satisfiability (SAT) problems, and to configure a commercial mixed integer programming solver.

Hyperopt

Bergstra et al. (2011) introduce Hyperopt, a software package that instead of modelling the performance of an algorithm given its hyperparameters, models the probability that the performance lies between a fixed quantile of the losses observed so far. These quantiles are in turn modeled with a Tree Parzen estimator, which also allows for efficient closed-form interpolation of expected hyperparameter performance.

Spearmint

Snoek et al. (2012) implement Bayesian optimization to sequentially train machine learning models under different hyperparameters. Spearmint assumes that the outputs of the unknown function (the performance of the machine learning model) correspond to samples from a Gaussian process. Thus, it is possible to estimate a posterior of the parameter distribution and, thanks to the Gaussian assumption, finding the maximum of the surrogate function is tractable.

5.1.1 Choosing a hyperparameter optimizer

Eggensperger et al. (2013) perform a head-to-head empirical comparison between SMAC, Hyperopt, and Spearmint on a series of benchmarks including MNIST classification (LeCun et al., 1998), online latent Dirichlet allocation (Blei et al., 2003) for Wikipedia articles, neural networks, and the Auto-Weka framework (Thornton et al., 2013). For a fixed running time or number of function evaluations, SMAC found better results than its competitors in 12 out of 15 scenarios. Based on this empirical evidence, we have decided to use SMAC for our experiments.

5.2 Experimental setup

We automatically optimize the hyperparameters of our algorithms using a Python implementation of SMAC, which calls our own routines implemented in Julia. After multiple iterations, SMAC returns the best hyperparameter configuration it found (also called the "incumbent" configuration), and the mean value that its internal model predicts for the best configuration.

We list the parameters that we expose to SMAC in Table 5.1. We also enforce a similar running time across different hyperparameter configurations by setting the number of ICM iterations to $\lfloor 32/\text{Number of ILS iterations} \rfloor$ (since in the default configuration $\lfloor 32/8 \rfloor = 4$ ICM iterations). We let SMAC test 200 hyperparameter configurations in total, and we add the constraints that the Temperature schedule and the Temperature decay should only be set when either SR-C or SR-D are chosen as the Stochastic eelaxation (SR) method. We initialize the codes B and the codebooks C with Optimized product quantization (OPQ) followed by Optimized tree quantization (OTQ). Finally, we perform 25 alternating updates of the codes B

Hyperparameter	Type	Domain	Default
Number of ILS iterations	Integer	$\{1, 2, 3, \dots 16\}$	8
Number of codebooks to perturb	Integer	$\{0, 1, 2, \dots m\}$	4
Randomize the ICM codebook order	Boolean	[true, false]	true
SR method	Categorical	{LSQ, SR-D, SR-C}	SR-D
Temperature schedule (Eq. 4.7-4.9)	Categorical	{1st, 2nd, 3rd}	1st
Temperature decay	Continuous	$[0, 1]$	0.5

Table 5.1: MCQ hyperparameters optimized by SMAC; m corresponds to the number of codebooks (7 or 15, depending on the memory budget). The default values correspond to the configuration that we have used in all our experiments so far.

and the codebooks C.

For datasets with *train/query/base* partitions, we encode the base set with the same number of ILS iterations as given by SMAC, and set the total number of ICM iterations to $\lfloor 128/\text{Number of ILS iterations} \rfloor$. This computational budget is similar to that used for the results reported in Chapter 3.

5.3 Results

For each dataset, we report the best configuration found by SMAC. We then run this configuration 10 times, and report the obtained recall@1. If under this scenario the new configuration is consistently better than the default configuration, we also increase the computational budget to 100 updates of B and C, and, for *train/query/base datasets*, encode the base set for $\lfloor 128/\text{Number of ILS iterations} \rfloor$ ICM iterations, which roughly corresponds to computational budget of the results reported in Chapter 4.

Configurations found by SMAC

We show the configurations obtained by SMAC in Table 5.2.

We note that SMAC always chooses to perturb at least one code, which is in line with our finding from Chapter 3 that adding perturbations to the local search routine is necessary to escape local minima. SMAC also always prefers SR-D or

Hyperparameter	LabelMe 64 bits	LabelMe 128 bits	MNIST 64 bits	MNIST 128 bits
Number of ILS iterations	9	8	9	8
Number of codebooks to perturb	1	4	5	4
Randomize the ICM codebook order	true	true	false	false
SR method	SR-D	SR-D	SR-D	SR-D
Temperature schedule (Eq. 4.7-4.9)	1st	1st	1st	1st
Temperature decay	0.43	0.5	0.19	0.83

Hyperparameter	SIFT1M 64 bits	SIFT1M 128 bits	Deep1M 64 bits	Deep1M 128 bits	VGG 64 bits	VGG 128 bits
Number of ILS iterations	8	7	8	15	8	10
Number of codebooks to perturb	4	2	4	2	4	5
Randomize the ICM codebook order	true	true	true	true	true	false
SR method	SR-D	SR-D	SR-D	SR-C	SR-C	SR-C
Temperature schedule (Eq. 4.7-4.9)	1st	1st	1st	1st	1st	1st
Temperature decay	0.65	0.19	0.5	0.95	0.71	0.94

Table 5.2: MCQ hyperparameters obtained by SMAC on query/base datasets (top) and train/query/base datasets (bottom).

SR-C over LSQ, which confirms our intuition from Chapter 4 that SR is always better than plain LSQ. Finally, SMAC also suggests the use of Schedule 1 (Eq 4.7) over the other temperature decay options; this is in line with our own findings, and those of Zeger et al. (1992).

Other parameter settings are more surprising. For example, SMAC has dropped the randomization of ICM on MNIST, both for 64 and 128 bits, and for VGG with 128 bits. The temperature decay is also more pronounced ($p > 0.9$) on the Deep1M and VGG datasets. Finally, there is ample variety in the number of ILS iterations including $\{7, 8, 9, 10, 15\}$, as well as in the number of codes to perturb: $\{1, 2, 4, 5\}$.

Testing the new configurations

We show the results obtained by the SMAC configurations and LSQ, SR-D and SR-C in Tables 5.3, 5.4, 5.5, 5.6, and 5.7. We omit values for SMAC when it did not suggest a different configuration from the default one.

On MNIST (Table 5.3) at 64 bits, the configuration found by SMAC obtains a 1% absolute improvement over the previous best results obtained by SR-D. We

		MNIST – 64 bits			MNIST – 128 bits		
		R@1	R@10	R@100	R@1	R@2	R@5
25 updates	LSQ-8	41.88 ± 0.28	93.60 ± 0.20	99.99 ± 0.01	63.26 ± 0.41	82.12 ± 0.50	96.28 ± 0.20
	SR-D-8	$\underline{42.30} \pm 0.33$	$\underline{93.74} \pm 0.15$	$\underline{99.99} \pm 0.01$	$\underline{63.65} \pm 0.28$	$\underline{82.41} \pm 0.32$	$\underline{96.41} \pm 0.20$
	SR-C-8	41.80 ± 0.36	93.55 ± 0.27	99.99 ± 0.01	61.36 ± 0.27	80.43 ± 0.24	95.48 ± 0.21
	SMAC	$\mathbf{43.62} \pm 0.41$	$\mathbf{94.36} \pm 0.14$	$\mathbf{100.00} \pm 0.01$	$\mathbf{64.09} \pm 0.29$	$\mathbf{82.81} \pm 0.23$	$\mathbf{96.61} \pm 0.20$
100 updates	LSQ-8	42.38 ± 0.49	93.82 ± 0.17	99.99 ± 0.01	64.93 ± 0.40	83.60 ± 0.35	96.91 ± 0.13
	SR-D-8	$\underline{42.90} \pm 0.34$	$\underline{93.83} \pm 0.27$	99.99 ± 0.01	$\underline{65.67} \pm 0.31$	$\underline{84.33} \pm 0.21$	$\underline{97.15} \pm 0.09$
	SR-C-8	42.26 ± 0.26	93.53 ± 0.28	99.99 ± 0.01	61.18 ± 0.38	80.26 ± 0.32	95.33 ± 0.20
	SMAC	$\mathbf{45.28} \pm 0.53$	$\mathbf{95.31} \pm 0.22$	$\mathbf{100.00} \pm 0.01$	$\mathbf{65.81} \pm 0.39$	$\mathbf{84.49} \pm 0.17$	$\mathbf{97.21} \pm 0.13$

Table 5.3: Detailed recall@N values on the MNIST dataset.

further investigate this result by running the configuration for 100 iterations – the computational budget of the results presented in Chapter 4. This leads to a total improvement of 2.38% (absolute) on this dataset, which defines a new state of the art. At 128 bits, the improvements are smaller – between 0 and 1% – yet consistent.

On the remaining datasets, the configurations obtained by SMAC do not consistently improve upon the default baseline. This suggests that the internal model of SMAC may have modelled noise in recall@N as an improvement.

		LabelMe – 64 bits			LabelMe – 128 bits		
		R@1	R@10	R@100	R@1	R@2	R@5
25 updates	LSQ-8	35.66 ± 0.71	84.32 ± 0.76	99.76 ± 0.09	53.73 ± 0.64	72.05 ± 0.49	89.81 ± 0.38
	SR-D-8	$\mathbf{36.64} \pm 0.69$	$\mathbf{85.23} \pm 0.56$	$\underline{99.78} \pm 0.09$	$\mathbf{54.51} \pm 0.86$	$\mathbf{72.78} \pm 0.92$	$\mathbf{90.70} \pm 0.67$
	SR-C-8	34.04 ± 1.67	82.15 ± 0.86	99.54 ± 0.12	44.54 ± 0.90	60.91 ± 1.23	81.38 ± 1.17
	SMAC	$\underline{36.40} \pm 0.94$	$\underline{85.16} \pm 0.49$	$\mathbf{99.80} \pm 0.11$	–	–	–

Table 5.4: Detailed recall@N values on the LabelMe dataset.

		SIFT1M – 64 bits			SIFT1M – 128 bits		
		R@1	R@10	R@100	R@1	R@2	R@5
25 updates	LSQ-32	29.42 ± 0.34	72.43 ± 0.43	97.36 ± 0.17	$\mathbf{55.15} \pm 0.34$	$\underline{72.14} \pm 0.43$	88.86 ± 0.29
	SR-D-32	$\mathbf{29.98} \pm 0.36$	72.87 ± 0.33	$\mathbf{97.47} \pm 0.15$	54.96 ± 0.64	72.05 ± 0.49	$\mathbf{88.98} \pm 0.17$
	SR-C-32	29.55 ± 0.45	$\underline{72.89} \pm 0.41$	97.40 ± 0.18	52.24 ± 0.47	69.15 ± 0.28	86.56 ± 0.21
	SMAC	$\underline{29.73} \pm 0.19$	$\mathbf{72.96} \pm 0.31$	$\underline{97.43} \pm 0.12$	$\underline{55.11} \pm 0.30$	$\mathbf{72.30} \pm 0.34$	$\underline{88.94} \pm 0.24$

Table 5.5: Detailed recall@N values on the SIFT1M dataset.

		Deep1M – 64 bits			Deep1M – 128 bits		
		R@1	R@10	R@100	R@1	R@2	R@5
25 updates	LSQ-32	21.11 ± 0.17	63.56 ± 0.49	94.81 ± 0.16	39.24 ± 0.49	55.48 ± 0.49	76.04 ± 0.34
	SR-D-32	21.33 ± 0.43	63.94 ± 0.30	95.02 ± 0.21	39.88 ± 0.48	56.33 ± 0.44	76.76 ± 0.45
	SR-C-32	**21.43** ± 0.22	**64.31** ± 0.48	**95.15** ± 0.21	**40.07** ± 0.33	56.41 ± 0.37	**77.02** ± 0.39
	SMAC	–	–	–	40.02 ± 0.43	**56.65** ± 0.46	76.80 ± 0.30

Table 5.6: Detailed recall@N values on the Deep1M dataset.

		VGG – 64 bits			VGG – 128 bits		
		R@1	R@10	R@100	R@1	R@2	R@5
25 updates	LSQ-32	17.63 ± 0.33	59.22 ± 0.48	95.85 ± 0.14	38.43 ± 0.55	54.91 ± 0.37	76.26 ± 0.44
	SRD-32	18.52 ± 0.38	60.86 ± 0.45	96.60 ± 0.26	39.19 ± 0.38	55.95 ± 0.34	76.84 ± 0.36
	SRC-32	**18.56** ± 0.43	**61.16** ± 0.48	96.63 ± 0.25	**42.13** ± 0.32	**59.47** ± 0.34	**80.81** ± 0.46
	SMAC	18.33 ± 0.24	60.82 ± 0.45	**96.65** ± 0.18	41.42 ± 0.38	58.48 ± 0.59	79.75 ± 0.46

Table 5.7: Detailed recall@N values on the VGG dataset.

5.4 Conclusions and future work

We have experimented with an automatic approach to algorithm configuration for MCQ. Using this method, we have found a new state-of-the-art configuration on MNIST, a dataset of raw images, but otherwise have obtained no improvement over the algorithm configuration that we had found by hand.

Our experiments were rather small-scale, as we were able to run them in less than a week on a standalone desktop with a single GPU. Future work may explore broader configurations (for example, including the OPQ initialization), or may let the optimization run for more iterative updates in each function probe.

Chapter 6

Deep stochastic local search: learning SLS perturbations with deep reinforcement learning

I stopped my dreaming
I won't do too much scheming these days
—Nico

In Chapter 3, we introduced a method based on iterated local search (ILS) for the encoding step of multi-codebook quantization (MCQ). Our encoding method is based on principles from stochastic local search (SLS) (Hoos and Stützle, 2004), a family of techniques based on random perturbations and iterative search known to work well for various NP-hard problems.

Drawing inspiration from general principles and designing new algorithms for specific problems is a typical approach to algorithm design. Doing so allows us to transfer knowledge across research fields, and gives us intuitive guiding principles to rely on during the design process.

An appealing alternative, however, is to avoid designing the encoding algorithm altogether, and instead learn a function that maps vectors to their assignments in a series of codebooks. Since the encoding problem is equivalent to maximum-a-posteriori (MAP) estimation in fully-connected Markov random fields (MRFs),

finding the optimal solution – even in a small set of vectors – is likely to be computationally demanding. In other words, acquiring labels (or "ground truth") to supervise the learning of an encoding function is inherently hard. Reinforcement learning (RL) (Sutton and Barto, 1998) provides a framework to learn functions without supervision, as long as we can provide a "reward" function.

However, learning an encoding function from scratch may prove too difficult with limited computation – MAP estimation is, after all, an NP-hard problem to which the research community has devoted many years of research. The same community has, at the same time, developed useful guiding algorithmic principles such as SLS. In this chapter, we ask: *is there a way in which we can combine the power of* SLS *algorithms and reinforcement learning*? We explore answers to this question in the context of learning algorithms for MCQ encoding.

6.1 Related work

Reinforcement learning (RL) is a classical subfield of artificial intelligence, distinct from the more common supervised learning in that RL can be used when no labels are available. The most prominent reference in the subject is the book by Sutton and Barto (1998).

In the most common setup, we have both an environment and an agent. Formally, the environment is modelled as a set of states $s \in S$, and the agent may perform a set of actions $a \in A$. The environment evolves stochastically, depending on both the current state and the action that the agent takes, which under a Markovian assumption can be fully described with a transition probability matrix $P(s' \mid s, a)$. Associated with each state-action pair, there is a (potentially latent) reward $R : S \times A \to \mathbb{R}$ that the agent obtains after each action. The goal is to learn a distribution over actions given the state (a.k.a. a "policy") $\pi(a \mid s)$ that maximizes reward over a series of actions.

6.1.1 Stochastic local search (SLS)

Stochastic local search (SLS) (Hoos and Stützle, 2004) defines a family of methods that alternate between searching the neighbourhood of a solution and stochastically perturbing the solution to escape local minima. Prominent examples of

SLS algorithms include Markov Chain Monte Carlo (MCMC), Simulated Annealing (which we used in Chapter 4) and Iterated local search (ILS) (Lourenço et al., 2003), (which we used in Chapter 3).

A commonality of SLS algorithms is that they all, at some point in the search process, sample from a distribution to perturb the current solution. In many cases, this distribution is either uniform, or otherwise amenable to parameterization and manipulation, such as a conjugate prior. We note that this distribution could instead be learned using, for example, a neural network. Furthermore, in a deep RL setting, this distribution could be seen as a "policy", and a pre-designed SLS algorithm could be seen as the "environment". This effectively allows us to leverage prior knowledge from SLS algorithms, incorporating an automatic learning component.

6.1.2 Deep reinforcement learning

RL provides a very general formulation that can in principle be applied to a wide range of problems. Classically, the agent would perceive the world via a set of handcrafted features. These features, however, have recently been replaced by deep neural networks that can be trained end-to-end for the task at hand. The intersection between deep learning and RL is often called "deep reinforcement learning" (or deep RL), and is our main subject of study in this chapter. Prominent breakthroughs of deep RL include attaining human-level performance in Atari games (Mnih et al., 2015), and superhuman performance in chess and the game of Go (Silver et al., 2017).

Since deep neural networks are typically trained with stochastic gradient descent (SGD) and the agent often makes a discrete action in the environment, a large body of work in deep RL focuses on methods to approximate gradients to train neural networks in discrete settings (Bengio et al., 2013; Sutton et al., 2000; Williams, 1992).

Visual attention

In modern convolutional neural networks (CNNs), the computational cost of processing an image is proportional to the image resolution. However, in tasks such as classification, the object of interest is often located in a relatively small part of

the image. Moreover, in tasks such as image captioning, where the model predicts one word at a time, it is common for different words to refer to different objects in the image. Xu et al. (2015) model the visual processing in image captioning as a convolutional network, and predict words using an Long Short-Term Memory unit (LSTM) (Hochreiter and Schmidhuber, 1997). To model visual attention, the LSTM selects a square region in the image at each prediction timestep. Since selecting a single region of an image is a non-differentiable operation, the authors use the REINFORCE algorithm (Williams, 1992) with multiple samples to estimate the gradient for backpropagation.

Optimizing language metrics

Rennie et al. (2017) notice that most metrics of language prediction, often borrowed for image captioning, are non-differentiable, and thus models are often trained on surrogate losses such as cross entropy. To bridge this gap between objectives, the authors use self-critical learning, a variant of the REINFORCE algorithm where the magnitude of the gradient is normalized with respect to the current prediction of the model, instead of a randomly sampled baseline. Crucially, the authors pre-train their models with a cross-entropy loss, and after supervised training plateaus they switch to RL training.

Superoptimizing programs

Bunel et al. (2017) apply deep RL to learn high-performance assembly programs for problems from the "hacker's delight" microbenchmarks (Warren, 2013). The authors define the reward function as a linear combination of speed and correctness over a large collection of unit tests. The neural network takes as input a bag-of-words representation of the output of the GCC compiler, and as output predicts a change to a candidate program. After multiple iterations and proposals, the final program obtained by the network is the output of the algorithm. This is very similar to a Monte-Carlo search, where the proposals are guided by the policy that the network learns. In RL terms, the space of programs is the set of states S, the actions A are changes to the current program, and the policy is the distribution over changes that the network proposes.

6.2 Problem formulation

Although the MCQ formulation contains two latent variables (the codebooks C and the codes B), we remind the reader that updating C is fairly easy; as in Chapter 4, we demonstrated that this can be done in less than a second for up to one million vectors. The main challenge lies in updating the codes B. For this reason, we assume that the codebooks are given in our formulation, and focus on learning a function

$$f^* = min_f \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}\left(f(\mathbf{x}_i, \theta), C\right), \tag{6.1}$$

where $\mathscr{L}$ is a loss function, and f has two main components:

1. A neural network with learnable parameters θ that predicts a series of distributions over codes $f(\mathbf{x}, \theta) \rightarrow p(\hat{\mathbf{b}}|\mathbf{x}, \theta)$. Since these are probability distributions, their components sum to one: $\sum_{k=1}^{h} \hat{\mathbf{b}}_{j,k} = 1$ for all j.

2. An SLS-based method that uses these distributions in the perturbation step, and produces a final set of codes $[\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m]$. The intuition is that these distributions should be learned such that they accelerate the finding of good solutions by SLS algorithms. For this reason, we refer to our approach as *deep stochastic local search*.

In our case, we experiment with several architectures for the first part of the function, and for the SLS method we use the encoding algorithm based on ILS with ICM encoding that we introduced in Chapter 3. The loss function – its negative being the reward – is simply the quantization error obtained by stochatic local search after using the proposed distributions and running for a number of iterations: $\mathscr{L}\left(f(\mathbf{x}_i, \theta), C\right) = \|\mathbf{x} - \sum_j^m C_j \cdot \mathbf{b}_j\|_2^2$.

6.3 Experimental setup

In our setup, we assume that a set of codebooks fully determines the set of states $s \in S$, as they define the unary and binary terms of the MRF. An action a is the prediction of a distribution over the set of codes, which our SLS algorithm samples from to compute quantization error – the negative of the "reward" $R(s,a)$. The

objective is to learn to predict a sampling distribution given the states – a "policy" $\pi(a \mid s)$ that the SLS algorithm can use to obtain good results faster than with a uniform distribution, as is typically used in SLS.

Datasets

We experiment on the train/query/base datasets that we used in Chapters 3 and 4: SIFT1M, VGG and Deep1M. We explicitly avoid query/base datasets such as LabelMe and MNIST because they lack a separate training partition. We learn our functions on the training sets, and test for generalization on the base set. Our goal is to improve recall@N for a fixed computational encoding budget.

Network designs

We experimented with three main network architecture:

1. An early fusion network, which has a small set of shared parameters before branching off to predict each codebook independently. The network is depicted in Figure 6.1 (a).

2. A late fusion network, which shares most of the parameters and branches off late to predict each codebook. This network is depicted in Figure 6.1 (b).

3. A recurrent neural network, which predicts each code sequentially, depicted in Figure 6.2.

Our feed-forward architectures are based on the bilinear block, a computationally efficient building block that has recently achieved state-of-the-art performance on 3d human pose estimation (Martinez et al., 2017b). Similarly, our recurrent architecture is based on a residual block that has shown state of the art performance on human motion prediction (Martinez et al., 2017a).

Training

We train our networks approximating the gradient with 32 runs of LSQ, and let each episode run for 8 ILS iterations. We experimented with both self-critical learning (Rennie et al., 2017) (*i.e.*, using the current prediction of the model to

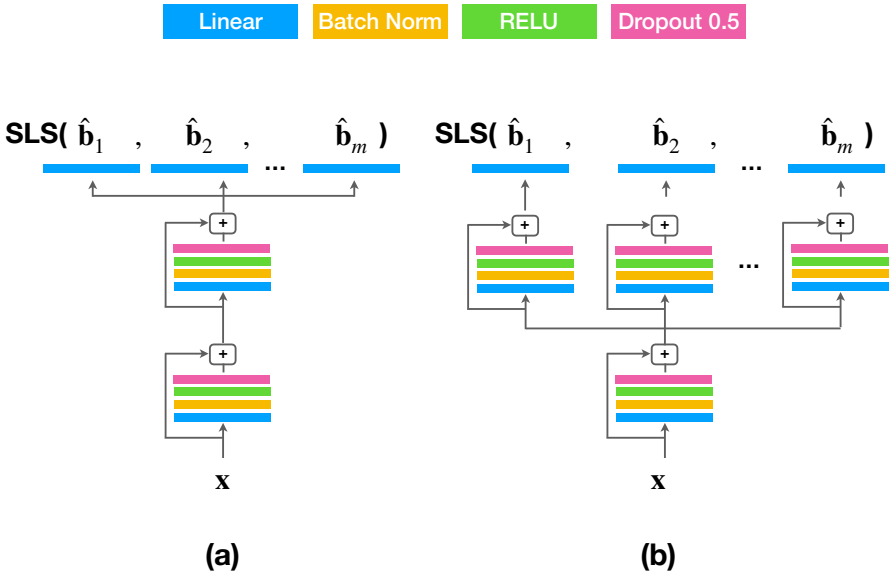

Figure 6.1: (a) Architecture with early fusion. (b) Architecture with late fusion.

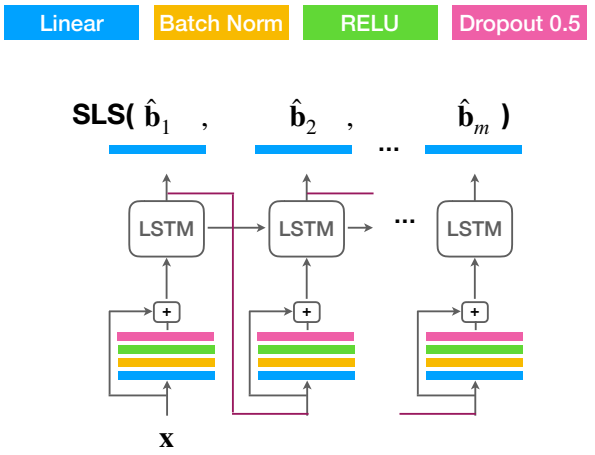

Figure 6.2: Recurrent architecture.

normalize the reward) and a random sampling baseline (as is common in RL), and found no difference in performance. We also experimented with the three architectures mentioned above, and obtained the best performance with the early fusion model.

We implemented our networks on Tensorflow (Abadi et al., 2016), and trained them for 4 epochs with Adam (Kingma and Ba, 2014), and a learning rate of 0.001 and a linear decay of 0.99 – this takes around one day of computation, and these parameters have shown good results on similar setups (Martinez et al., 2017b).

6.4 Results

Since we are interested in accelerating the optimization speed of LSQ encoding, we compare vanilla LSQ, SR-D and SR-C (that is, the methods sampling from uniform distributions), vs the same methods sampling from learned distributions. The results for our main network are shown on Figure 6.3.

We observe that learning a sampling distribution barely makes a difference in most cases. The only exception is on the VGG dataset at 128 bits, where SR-C with a uniform distribution has a poor start, and catches up with other methods at 16 ILS iterations to finally outperform the rest of the baselines by rougly 0.05 in recall. In this case, learning a sampling distribution accelerates convergence during the first 16 iterations, but progress is roughly equal after 32 iterations.

6.5 Conclusions and future work

We have shown our attempts to learn encoding functions for MCQ combining Stochastic local search (SLS) with deep reinforcement learning. We have described the learning algorithms and network architectures that we used, an discussed the limitations of our approach. In particular, the functions that our models learned did not noticeably improve upon an SLS baseline with a uniform (not learned) distribution in most cases, and never did so after 16 ILS iterations.

Multiple reasons could be to responsible for the shortcomings of our approach in experimental performance. For once, MAP estimation might simply be a hard problem for a neural network to solve, as we are not aware of deep RL outperforming classical algorithms in similar scenarios. Another explanation might be

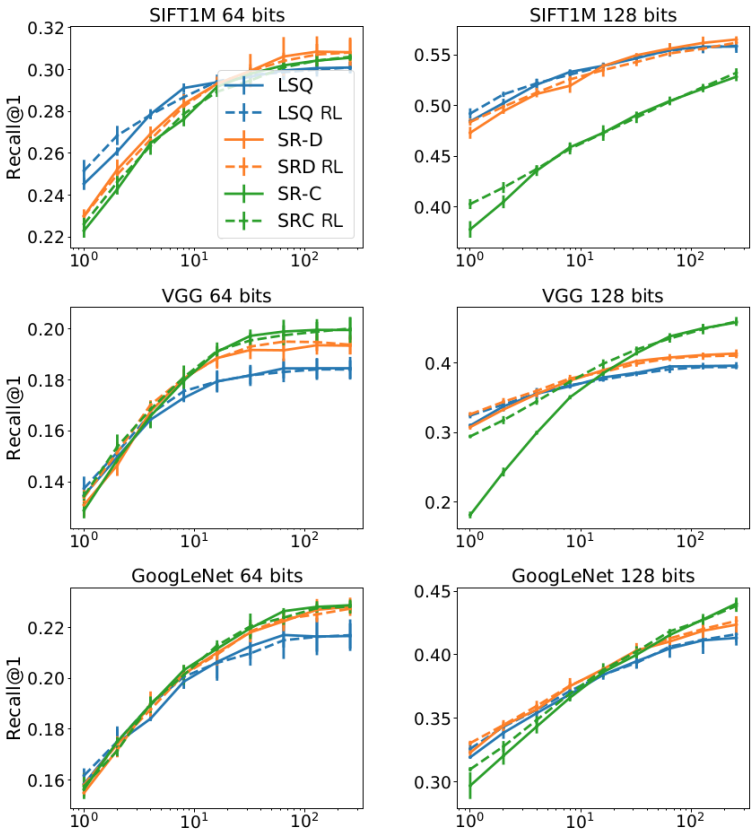

Figure 6.3: Recall@1 as a function of ILS iterations on the SIFT1M, VGG and Deep1M datasets. The methods with "RL" are using learned sampling distributions, while those without "RL" correspond to the methods introduced in Chapters 3 and 4 (ie, using a uniform distribution for perturbation sampling).

that we simply need larger models and more computation to robustly learn sampling distributions; after all, recent breakthroughs in deep RL (Mnih et al., 2015; Silver et al., 2017) have used several orders of magnitude more computation than is currently available to us. Revisiting these approaches in the future, with more available compute cycles and more robust deep RL algorithms, is an interesting area of future work.

Chapter 7

Rayuela.jl: A package for large-scale similarity search

The paramedic thinks I'm clever cause I play guitar
I think she's clever cause she stops people dying
—Courtney Barnett

Together with this thesis we are releasing Rayuela.jl, a software package for large-scale similarity search written in Julia (Bezanson et al., 2014). Rayuela.jl is available under an MIT licence, and can be downloaded at https://github.com/una-dinosauria/Rayuela.jl. Multiple motivations led us to develop our library:

- **As an approximate nearest neighbour search (ANNS) tool**. In practice, approximate nearest neighbour search (ANNS) is rarely a problem of interest by itself. In other words, ANNS typically arises as a subproblem of other task such as large-scale retrieval or classification. We want to release a library that developers can use to tackle large-scale ANNS problems that they run into, therefore maximizing the impact of our research.

- **As a free alternative to FAISS**. Facebook AI Similarity Search (FAISS) (Johnson et al., 2017) is another library that implements multiple MCQ baselines. Unfortunately, FAISS currently implements only orthogonal MCQ methods. Furthermore, FAISS is distributed under a BSD + Patents licence, which

includes strong patent retaliation terms. This licence has been explicitly disavowed by the Apache Software Foundation[1], which prompted multiple open source software projects to stop using React.js, a popular javascript library written by Facebook. In September 2017, Facebook changed the licence of React.js and other of their projects to MIT, but to date has not changed the licence of FAISS.

- **For scientific reproducibility**. We also want to provide a free codebase where known MCQ baselines can be reproduced, and reference implementations can be freely consulted. This has the side effect of making our research reproducible, which is unfortunately not common in this field (yet).

- **To test Julia's capabilities**. Julia is a nascent programming language that is emerging as an alternative to Python and Matlab in scientific computing. Writing our implementations in Julia serves as a use-case from which we derived insights into the advantages and limitations of this new language.

There are multiple benefits of reproducible research beyond ensuring the integrity of publicly funded research. We find the following note by Donoho (2010) particularly eloquent:

> In my own experience, error is ubiquitous in scientific computing, and one needs to work very diligently and energetically to eliminate it. One needs a very clear idea of what has been done in order to know where to look for likely sources of error. I often cannot really be sure what a student or colleague has done from [their] own presentation, and in fact often [their] description does not agree with my own understanding of what has been done, once I look carefully at the scripts. Actually, I find that researchers quite generally forget what they have done and misrepresent their computations.
>
> Computing results are now being presented in a very loose, breezy way in journal articles, in conferences, and in books. All too often one simply takes computations at face value. This is spectacularly against

[1] https://github.com/facebook/react/issues/10191

> the evidence of my own experience. I would much rather that at talks and in referee reports, the possibility of such error were seriously examined.

In the following, we describe the functionality implemented in Rayuela.jl. We then describe some of the strengths and downsides of Julia as our programming language of choice, and finally discuss future work that we would like to see implemented in Rayuela.jl.

7.1 Functionality

The largest effort, parallel to ours, to reproduce MCQ baselines is currently the Facebook AI Similarity Search (FAISS) library (Johnson et al., 2017). However, FAISS currently does not implement any non-orthogonal MCQ methods – Rayuela.jl fills this gap in reproducible research. In Table 7.1 we list a number of MCQ algorithms, whether the authors released code with their paper, whether the algorithm is implemented in FAISS, and whether it is implemented in Rayuela.jl. Overall, Rayuela implements seven MCQ methods that FAISS does not currently provide.

Rayuela.jl, like FAISS, includes GPU-accelerated versions of the most computationally-expensive algorithms that it implements. In our case, we provide CUDA implementations of LSQ and SR (both introduced in this thesis), and OTQ (Babenko and Lempitsky, 2015).

7.2 Strengths

The main motivation behind developing Julia as a language for scientific computing is that languages such as R, Python and Matlab are interpreted, not compiled. While interpreters are good for rapid prototyping, they are often not good at generating efficient low-level instructions. In practice, this means that users rely on libraries written in C (such as Numpy), and often have to write their own performance-critical routines in lower-level languages to achieve satisfactory running times. This issue is commonly known as the "two language problem". Julia leverages the Low Level Virtual Machine (LLVM) compiler infrastructure with a strong type system, which allows it to produce highly efficient machine code. This,

Name (Reference)	Abbr.	Code	FAISS (GPU)	Rayuela (GPU)
Orthogonal methods				
Transform coding (Brandt, 2010)	TC			
Expectation-based coding (Sandhawalia and Jégou, 2010)	–			
Product quantization (Jégou et al., 2009, 2011)	PQ	Matlab, C	✓(✓)	✓
Optimized product quantization (Ge et al., 2014)	OPQ	Matlab	✓(✓)	✓
Cartesian *k*-means (Norouzi and Fleet, 2013)	CKM	Matlab, C++	✓(✓)	✓
Polysemous codes (Douze et al., 2016)	–	C++	✓	
Semi-orthogonal methods				
Composite quantization (Zhang et al., 2014)	CQ	MSVC C++		
Optimized tree quantization (Babenko and Lempitsky, 2015)	OTQ			✓(✓)
Optimized Cartesian *k*-means (Wang et al., 2014)	OCKM			
Non-orthogonal methods				
Residual vector quantization (Chen et al., 2010)	RVQ			✓
Enhanced residual vector quantization (Ai et al., 2014, 2017)	ERVQ			✓
Additive quantization (Babenko and Lempitsky, 2014a)	AQ	Python		
Stacked quantizers (Martinez et al., 2014)	SQ	Matlab, C++		✓
Self-organizing binary encoding (Ozan et al., 2016c)	SOBE			
Joint *k*-means quantization (Ozan et al., 2016b)	JKM			
Competitive quantization (Ozan et al., 2016a)	COMPQ			✓
Pyramid additive quantization (Muravev et al., 2017)	–			
Fast additive quantization (Li et al., 2017)	–			
Local search quantization (This thesis)	LSQ	Julia, CUDA		✓(✓)
Stochastic eelaxations (This thesis)	SR	Julia, CUDA		✓(✓)

Table 7.1: Summary of MCQ methods and their implementations

in theory, allows users to write their entire codebases in Julia, without having to resort to lower-level languages.

In our experience, Julia delivers on this promise. While we provide C++ implementations of both LSQ and OTQ encoding, we note that the Julia version of LSQ is only 1.5-2× slower than the C++ version, and the Julia version of OTQ encoding is in fact faster and more memory-efficient than the C++ version.

7.3 Challenges and limitations

Most drawbacks of Julia as a language are performance issues. Before the Julia 1.0 release, language developers have been focused on implementing language features, and some routines are lacking in speed with respect to their C/C++ coun-

terparts. We explain some instances and consequences of these drawbacks in detail.

MCQ search code: slow sorting and (lack of) multithreading

The search code takes a query vector $\mathbf{q}$ and computes distance tables with respect to the codebooks C. It then uses table lookups to compute approximate distances to the compressed database vectors B and, at the end, partially sorts the distances to produce a final ranking of the top k neighbours. Our search code is currently written in C++. The two main reasons for this are (1) that sorting in Julia is slow, and (2) that multithreading in Julia is not fully supported.

Slow sorting (issue #939) was first reported in 2012 and, to date, Julia's performance remains about $2\times$ slower than its C++ counterpart.

Unfortunately, to this day multithreading is only supported as an experimental feature in Julia. Historically, this is due to LLVM (the compiler infrastructure that Julia uses) not originally supporting OpenMP. LLVM started supporting OpenMP development in 2014, but by then Julia was a rather mature language, conceived without parallelism at its core. Slow sorting and lack of multithreading make our C++ implementation search code about $6\times$ faster in C++ than in Julia, so we choose to keep the C++ implementation in the code base for the time being.

ICM encoding: compiling Julia from source

Iterated conditional modes (ICM) with random perturbations is the core algorithm inside Local search quantization (LSQ), introduced in Chapter 3. Since the algorithm may run on up to billions of vectors, it was crucial for us to develop a fast implementation. We have developed a CUDA implementation of this algorithm, and we also provide a C++ implementation that is 1.5-$2\times$ faster than the Julia baseline.

Unfortunately, running CUDA code in Julia currently requires the user to compile Julia from source. This is sometimes non trivial and error-prone, and not ideal from a usability perspective.

7.4 Future work

Future work should address the limitation that we have described. Ideally, we should be able to provide a package that is fully written in Julia, that offers multi-threading support, and that has a speed comparable to a C implementation.

To the best of our knowledge, mulithreading will be officially supported in a future version of Julia (1.x). When such support comes, we would like to use the feature to eliminate the remaining the C++ code from our package.

Getting rid of sorting in search code

Since in ANNS the user often needs only a list of k nearest neighbours, we could use a heap to keep track of the top k nearest neighbours at any given time, instead of sorting them in the end. If the implementation of this data structure is efficient enough, we may be able to bypass the slow sorting issue and eliminate the C++ search code.

Writing CUDA code in Julia

There is an interesting initiative to generate CUDA code from Julia called CUDA-native.jl (Besard et al., 2017). CUDANative.jl implements currently only a subset of CUDA functionality, but this might be enough to generate the LSQ and OTQ encoding routines that we currently provide in CUDA. The Julia 1.0 release should drop the requirement for users to compile Julia from source.

More MCQ methods

We would like to implement other MCQ methods which, to this day, lack a public implementation and are therefore hard to reproduce. The method by Guo et al. (2016) runs k-means under the Mahalanobis distance to learn a compression amenable to maximum inner product search (MIPS). Another interesting approach, due to Wu et al. (2017), is based on Stochatic gradient descent and focuses on distributions with large range of norms, such as those arising in inverted files. It would be interesting to compare these two approaches with our current library to empirically determine their performance against state-of-the-art MCQ approaches.

7.5 Conclusion

We have introduced Rayuela.jl, a package for large-scale similarity search. Rayuela fulfills our commitment to make our research accessible and reproducible, and fills an important gap in reproducible research in the MCQ literature. We have also discussed the strengths and limitations of Julia as a programming language of choice, and discussed future work that will make the package faster, more portable, and easier to use.

Chapter 8

Conclusion

Did I do okay?
Did I pave the way?
—Belle and Sebastian

In this thesis, we have introduced several improvements to algorithms that produce compact codes used in large-scale similarity search. In Chapter 3, we introduced a method for fast encoding with better computation vs accuracy trade-offs than its competitors. In Chapter 4, we added noise to either the codes or the codebooks with a decaying temperature schedule, resulting in improved solutions. We also introduced a method for fast codebook update that exploits the structure of the binary codes, resulting in shorter running times. These contributions make it more appealing to use non-orthogonal codebooks in MCQ, as they drastically reduce the computation-vs-accuracy trade-offs of current methods.

We have also used an automatic hyperparameter configuration tool (Hutter et al., 2011) to adapt our algorithms to particular data distributions in Chapter 5. This resulted in a new state of the art in a dataset of raw image digits (LeCun et al., 1998). Finally, we also explored the use of deep reinforcement learning techniques to learn sampling distributions for stochastic local search procedures in Chapter 6. This, unfortunately, did not lead to noticeable improvements for most datasets.

The main limitation that we faced during the development of our work was, largely, that it was very challenging for us to reproduce previous baselines. A significant step towards reversing this trend, we are releasing an open source library

that implements all our contributions and several MCQ baselines. Our implementation is detailed in Chapter 7.

Future work

We identify mainly three areas of future work in multi-codebook quatization (MCQ)

1. Improvements upon local-search quantization (LSQ), introduced in this work, may focus on reducing the running time of the code update step, which is its computational bottleneck. This means more research into algorithms for large-scale combinatorial problems.

2. Improvements upon composite quantization (CQ) (Zhang et al., 2014) may focus on reducing the running time of the codebook update step, which is currently its bottleneck. Currently, the algorithm uses the L-BFGS (Nocedal, 1980) algorithm to solve a constrained optimization problem. Exploring the use of recent algorithms for large-scale continuous optimization such as the stochastic average gradient (Schmidt et al., 2017) may prove useful in this case.

3. Finally, improvements upon competitive quantization (COMPQ) (Ozan et al., 2016a) may focus on reducing the overall running time of this gradient-based algorithm. With its gradient-based optimization, this algorithm is uniquely situated to be incorporated in

An interesting direction of future research in MCQ lies in incorporating vector compression into end-to-end trainable learning pipelines. The work by Jain et al. (2018) is a first step in this direction that backpropagates quantization error to the indexing structure. Their work, however, is technically based on hashing, as it computes distances with Hamming codes and does not use codebooks. Klein and Wolf (2017) explicitly learn codebooks and codes in a deep learning pipeline, thus introducing the first fully-differentiable MCQ pipeline. Merging these two ideas into a fully-differentiable, deeply-learned indexing structure is an interesting open problem in the field.

Bibliography

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016. → pages 21, 86

L. Ai, Y. Junqing, T. Guan, and Y. He. Efficient approximate nearest neighbor search by optimized residual vector quantization. In *Content-Based Multimedia Indexing (CBMI), 12th International Workshop on*, pages 1–4, 2014. → pages 15, 18, 38, 49, 60, 91

L. Ai, J. Yu, Z. Wu, Y. He, and T. Guan. Optimized residual vector quantization for efficient approximate nearest neighbor search. *Multimedia Systems*, 23(2): 169–181, 2017. → pages 15, 18, 91

A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt. Practical and optimal LSH for angular distance. In *Advances in Neural Information Processing Systems*, pages 1225–1233, 2015. → pages 9

F. André, A.-M. Kermarrec, and N. Le Scouarnec. Cache locality is not enough: high-performance nearest neighbor search with product quantization fast scan. *Proceedings of the VLDB Endowment*, 9(4):288–299, 2015. → pages 18, 19, 35

D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM Symposium on Discrete Algorithms*. → pages 48, 51, 52

F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991. → pages 7

A. Babenko and V. Lempitsky. The inverted multi-index. In *Computer Vision and Pattern Recognition*, pages 3069–3076, 2012. → pages 21, 35, 71

A. Babenko and V. Lempitsky. Additive quantization for extreme vector compression. In *Computer Vision and Pattern Recognition*, pages 931–938,

2014a. → pages 3, 4, 15, 16, 18, 22, 23, 24, 25, 26, 35, 37, 38, 40, 41, 50, 53, 91

A. Babenko and V. Lempitsky. Improving bilayer product quantization for billion-scale approximate nearest neighbors in high dimensions. *arXiv preprint arXiv:1404.1831*, 2014b. → pages 35, 40

A. Babenko and V. Lempitsky. Tree quantization for large-scale similarity search and classification. In *Computer Vision and Pattern Recognition*, pages 4240–4248, 2015. → pages 3, 14, 16, 18, 29, 31, 37, 38, 40, 90, 91

A. Babenko and V. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *Computer Vision and Pattern Recognition*, pages 2055–2063, 2016. → pages 59, 61, 66

A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky. Neural codes for image retrieval. In *European Conference on Computer Vision*, pages 584–599. Springer, 2014. → pages 39

Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. → pages 81

J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975. → pages 8

J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011. → pages 72, 73

T. Besard, C. Foket, and B. De Sutter. Effective extensible programming: Unleashing julia on GPUs. *arXiv preprint arXiv:1712.03112*, 2017. → pages 93

J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *arXiv preprint arXiv:1411.1607*, 2014. → pages 5, 46, 88

D. W. Blalock and J. V. Guttag. Bolt: Accelerated data mining with fast vector compression. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 727–735, 2017. → pages 19

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3(1):993–1022, 2003. → pages 74

Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001. → pages 26

J. Brandt. Transform coding for fast approximate nearest neighbor search in high dimensions. In *Computer Vision and Pattern Recognition*, pages 1815–1822, 2010. → pages 12, 18, 91

E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010. → pages 73

R. Bunel, A. Desmaison, M. P. Kumar, P. H. Torr, and P. Kohli. Learning to superoptimize programs. In *International Conference on Learning Representations*, 2017. → pages 82

Y. Cao, M. Long, J. Wang, H. Zhu, and Q. Wen. Deep quantization network for efficient image retrieval. In *Association for the Advancement of Artificial Intelligence*, pages 3457–3463, 2016. → pages 20

Y. Cao, M. Long, J. Wang, and S. Liu. Deep visual-semantic quantization for efficient image retrieval. In *Computer Vision and Pattern Recognition*, volume 2, pages 1328–1337, 2017. → pages 21

M. E. Celebi, H. A. Kingravi, and P. A. Vela. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications*, 40(1):200–210, 2013. → pages 51

W.-Y. Chan, S. Gupta, and A. Gersho. Enhanced multistage vector quantization by joint codebook design. *IEEE Transactions on Communications*, 40(11): 1693–1697, 1992. → pages 15

K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*, 2014. → pages 39, 59

Y. Chen, T. Guan, and C. Wang. Approximate nearest neighbor search by residual vector quantization. *Sensors*, 10(12):11259–11273, 2010. → pages 15, 16, 18, 23, 38, 49, 60, 67, 91

G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990. → pages 24

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition*, pages 248–255, 2009. → pages 2

D. L. Donoho. An invitation to reproducible computational research. *Biostatistics*, 11(3):385–388, 2010. → pages 89

M. Douze, H. Jégou, and F. Perronnin. Polysemous codes. In *European Conference on Computer Vision*, pages 785–801, 2016. → pages 18, 19, 21, 91

J. C. Dunn. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57, 1973. → pages 51

K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice*, volume 10, 2013. → pages 74

C. Elkan. Using the triangle inequality to accelerate k-means. In *International Conference on Machine Learning*, pages 147–153, 2003. → pages 51

J. K. Flanagan, D. R. Morrell, R. L. Frost, C. J. Read, and B. E. Nelson. Vector quantization codebook generation using simulated annealing. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 1759–1762, 1989. → pages 52

D. C.-L. Fong and M. Saunders. LSMR: An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing*, 33(5): 2950–2971, 2011. → pages 50, 53

T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. *Transactions on Pattern Analysis and Machine Intelligence*, 36(4):744–755, 2014. → pages 3, 12, 18, 37, 38, 39, 40, 60, 91

S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984. → pages 26

A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992. ISBN 0792391810;9780792391814;. → pages 24

Y. Gong and S. Lazebnik. Iterative quantization: A Procrustean approach to learning binary codes. In *Computer Vision and Pattern Recognition*, pages 817–824, 2011. → pages 3, 60

A. Gordo, J. Almazan, J. Revaud, and D. Larlus. End-to-end learning of deep visual representations for image retrieval. *International Journal of Computer Vision*, 124(2):237–254, 2017. → pages 2

R. Guo, S. Kumar, K. Choromanski, and D. Simcha. Quantization based fast inner product search. In *Artificial Intelligence and Statistics*, pages 482–490, 2016. → pages 3, 20, 93

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. → pages 82

H. H. Hoos and T. Stützle. *Stochastic local search: Foundations & Applications*. Elsevier, 2004. → pages 22, 27, 30, 42, 55, 79, 80

F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Association for the Advancement of Artificial Intelligence*, volume 7, pages 1152–1157, 2007. → pages 72, 73

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer, 2011. → pages 72, 73, 95

Y. Hwang, B. Han, and H.-K. Ahn. A fast nearest neighbor search algorithm by nonlinear embedding. In *Computer Vision and Pattern Recognition*, pages 3053–3060, 2012a. → pages 51

Y. Hwang, B. Han, and H.-K. Ahn. A fast nearest neighbor search algorithm by nonlinear embedding. In *Computer Vision and Pattern Recognition*, pages 3053–3060, 2012b. → pages 15

A. K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010. → pages 51

H. Jain, P. Pérez, R. Gribonval, J. Zepeda, and H. Jégou. Approximate search with quantized sparse representations. In *European Conference on Computer Vision*, pages 681–696, 2016. → pages 19

H. Jain, J. Zepeda, P. Pérez, and R. Gribonval. SUBIC: A supervised, structured binary code for image search. In *International Conference on Computer Vision*, pages 833–842, 2017. → pages 21

H. Jain, J. Zepeda, P. Pérez, and R. Gribonval. Learning a complete image indexing pipeline. *Computer Vision and Pattern Recognition*, pages 4933–4941, 2018. → pages 21, 96

M. Järvisalo, D. Le Berre, O. Roussel, and L. Simon. The international SAT solver competitions. *AI Magazine*, 33(1):89–92, 2012. → pages 22

H. Jégou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *European Conference on Computer Vision*, pages 304–317, 2008. → pages 11

H. Jégou, M. Douze, and C. Schmid. Searching with quantization: approximate nearest neighbor search using short codes and distance estimators. Technical Report RR-7020, INRIA, 2009. → pages 11, 12, 18, 91

H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *Transactions on Pattern Analysis and Machine Intelligence*, 33(1): 117–128, 2011. → pages 3, 12, 18, 21, 37, 38, 39, 40, 59, 60, 61, 67, 71, 91

H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Perez, and C. Schmid. Aggregating local image descriptors into compact codes. *Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1704–1716, 2012. → pages 12

J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734*, 2017. → pages 21, 88, 90

Y. Kalantidis and Y. Avrithis. Locally optimized product quantization for approximate nearest neighbor search. In *Computer Vision and Pattern Recognition*, pages 2321–2328, 2014. → pages 40

J. H. Kappes, B. Andres, F. Hamprecht, C. Schnorr, S. Nowozin, D. Batra, S. Kim, B. X. Kausler, J. Lellmann, N. Komodakis, et al. A comparative study of modern inference techniques for discrete energy minimization problems. In *Computer Vision and Pattern Recognition*, pages 1328–1335, 2013. → pages 25, 29

A. M. Kibriya and E. Frank. An empirical comparison of exact nearest neighbour algorithms. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 140–151, 2007. → pages 8

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. → pages 86

B. Klein and L. Wolf. In defense of product quantization. *arXiv preprint arXiv:1711.08589*, 2017. → pages 21, 96

P. Krähenbühl and V. Koltun. Efficient inference in fully connected CRFs with Gaussian edge potentials. In *Advances in Neural Information Processing Systems*, pages 109–117, 2011. → pages 26

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Computer Vision and Pattern Recognition*, pages 1097–1105, 2012. → pages 72

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. → pages 39, 59, 74, 95

J. Li, X. Lan, J. Wang, M. Yang, and N. Zheng. Fast additive quantization for vector compression in nearest neighbor search. *Multimedia Tools and Applications*, 76(22):23273–23289, 2017. → pages 17, 18, 91

Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transactions on communications*, 28(1):84–95, 1980. → pages 48, 52

S. Liu, J. Shao, and H. Lu. Generalized residual vector quantization for large scale data. 2016. → pages 16

H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In *Handbook of Metaheuristics*, pages 320–353. Springer, 2003. → pages 27, 81

D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. → pages 2, 10

Y. A. Malkov and D. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *arXiv preprint arXiv:1603.09320*, 2016. → pages 9

T. M. Martinetz, S. G. Berkovich, and K. J. Schulten. 'Neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, 1993. → pages 51

J. Martinez, H. H. Hoos, and J. J. Little. Stacked quantizers for compositional vector compression. *arXiv preprint arXiv:1411.2173*, 2014. → pages 15, 18, 20, 38, 49, 60, 91

J. Martinez, J. Clement, H. H. Hoos, and J. J. Little. Revisiting additive quantization. In *European Conference on Computer Vision*, pages 137–153, 2016a. → pages v

J. Martinez, H. H. Hoos, and J. J. Little. Solving multi-codebook quantization in the GPU. In *Workshop on Web-scale Vision and Social Media (VSM), ECCV workshops*, 2016b. → pages v

J. Martinez, M. J. Black, and J. Romero. On human motion prediction using recurrent neural networks. In *Computer Vision and Pattern Recognition*, pages 4674–4683, 2017a. → pages 84

J. Martinez, R. Hossain, J. Romero, and J. J. Little. A simple, yet effective baseline for 3d human pose estimation. In *International Conference on Computer Vision*, pages 2640–2649, 2017b. → pages 84, 86

J. Martinez, S. Zakhmi, H. H. Hoos, and J. J. Little. LSQ++: Lower running time and higher recall in multi-codebook quantization. In *European Conference on Computer Vision*, 2018. → pages v

Y. Matsui, T. Yamasaki, and K. Aizawa. PQtable: Fast exact asymmetric distance neighbor search for product quantization using hash tables. In *International Conference on Computer Vision*, pages 1940–1948, 2015. → pages 71

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. → pages 81, 87

M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009. → pages 8, 9, 12, 30

A. Muravev, E. C. Ozan, A. Iosifidis, and M. Gabbouj. Pyramid encoding for fast additive quantization. In *Signal Processing Conference*, 2017. → pages 17, 18, 91

Y. Nagata and S. Kobayashi. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *Informs Journal on Computing*, 25(2):346–363, 2013. → pages 27

D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Computer Vision and Pattern Recognition*, volume 2, pages 2161–2168, 2006. → pages 9

J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980. → pages 13, 96

M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *International Conference on Machine Learning*, pages 353–360, 2011. → pages 39

M. Norouzi and D. J. Fleet. Cartesian k-means. In *Computer Vision and Pattern Recognition*, pages 3017–3024, 2013. → pages 3, 13, 18, 39, 60, 91

M. Norouzi, A. Punjani, and D. J. Fleet. Fast exact search in hamming space with multi-index hashing. *Transactions on Pattern Analysis and Machine Intelligence*, 36(6), 2014. → pages 8

J. Nutini, M. Schmidt, I. Laradji, M. Friedlander, and H. Koepke. Coordinate descent converges faster with the gauss-southwell rule than random selection. In *International Conference on Machine Learning*, pages 1632–1641, 2015. → pages 30

A. Oliva and A. Torralba. Building the gist of a scene: The role of global image features in recognition. *Progress in Brain Research*, 155:23–36, 2006. → pages 17

S. M. Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989. → pages 8

E. C. Ozan, S. Kiranyaz, and M. Gabbouj. Competitive quantization for approximate nearest neighbor search. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2884–2894, 2016a. → pages 3, 17, 18, 40, 66, 67, 91, 96

E. C. Ozan, S. Kiranyaz, and M. Gabbouj. Joint k-means quantization for approximate nearest neighbor search. In *International Conference on Pattern Recognition*, pages 3645–3649, 2016b. → pages 16, 17, 18, 91

E. C. Ozan, S. Kiranyaz, M. Gabbouj, and X. Hu. Self-organizing binary encoding for approximate nearest neighbor search. In *Signal Processing Conference*, pages 1103–1107, 2016c. → pages 16, 17, 18, 91

G. Papandreou and A. Yuille. Perturb-and-MAP random fields: Using discrete optimization to learn and sample from energy models. In *International Conference on Computer Vision*, pages 193–200, 2011. → pages 68

J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Computer Vision and Pattern Recognition*, pages 1–8, 2007. → pages 51

A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops*, pages 512–519. IEEE, 2014. → pages 39

S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel. Self-critical sequence training for image captioning. In *Computer Vision and Pattern Recognition*, pages 7008–7016, 2017. → pages 82, 84

B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1-3):157–173, 2008. → pages 59

H. Sandhawalia and H. Jégou. Searching with expectations. In *International Conference on Acoustics Speech and Signal Processing*, pages 1242–1245, 2010. → pages 11, 12, 18, 91

M. Schmidt. UGM: Matlab code for undirected graphical models. http://www.cs.ubc.ca/~schmidtm/Software/UGM/, 2007. Accessed: 2018-08-18. → pages 30

M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017. → pages 69, 96

A. Shrivastava and P. Li. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Advances in Neural Information Processing Systems*, pages 2321–2329, 2014. → pages 3

C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008. → pages 9

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354, 2017. → pages 81, 87

J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *International Conference on Computer Vision*, pages 1–8, 2003. → pages 11

N. Snavely, S. M. Seitz, and R. Szeliski. 25(3):835–846, 2006. → pages 2

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012. → pages 72, 74

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014. → pages 72

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*, volume 1. MIT Press Cambridge, 1998. → pages 80

R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063, 2000. → pages 67, 81

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014. → pages 59

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 847–855, 2013. → pages 74

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. → pages 37

A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008. → pages 2

J. Vaisey and A. Gersho. Simulated annealing and codebook design. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 1176–1179, 1988. → pages 52

E. van den Berg and M. P. Friedlander. Probing the Pareto frontier for basis pursuit solutions. *SIAM Journal on Scientific Computing*, 31(2):890–912, 2008. → pages 37

V. Vineet and P. Narayanan. Cuda cuts: Fast graph cuts on the GPU. In *Computer Vision and Pattern Recognition Workshops*, pages 1–8, 2008. → pages 26, 27

J. Wang and T. Zhang. Composite quantization. *arXiv preprint arXiv:1712.00955*, 2017. → pages 13

J. Wang, J. Wang, J. Song, X.-S. Xu, H. T. Shen, and S. Li. Optimized Cartesian k-means. *IEEE Transactions on Knowledge and Data Engineering*, 27(1): 180–192, 2014. → pages 14, 16, 18, 91

X. Wang, T. Zhang, G.-J. Qi, J. Tang, and J. Wang. Supervised quantization for similarity search. In *Computer Vision and Pattern Recognition*, pages 2018–2026, 2016. → pages 20

H. S. Warren. *Hacker's Delight*. Pearson Education, 2013. → pages 82

Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems*, pages 1753–1760, 2009. → pages 3, 11

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992. → pages 81, 82

J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Computer Vision and Pattern Recognition*, pages 4820–4828, 2016. → pages 3

X. Wu, R. Guo, A. T. Suresh, S. Kumar, D. N. Holtmann-Rice, D. Simcha, and X. Y. Felix. Multiscale quantization for fast similarity search. In *Advances in Neural Information Processing Systems*, pages 5749–5757, 2017. → pages 20, 93

Y. Xia, K. He, F. Wen, and J. Sun. Joint inverted indexing. In *International Conference on Computer Vision*, pages 3416–3423, 2013. → pages 35

K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015. → pages 82

L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32: 565–606, 2008. → pages 22

C. Zach, D. Gallup, J.-M. Frahm, and M. Niethammer. Fast global labeling for real-time stereo using multiple plane sweeps. In *Vision, Modelling and Visualization (VMV)*, pages 243–252, 2008. → pages 26, 27

K. Zeger, J. Vaisey, and A. Gersho. Globally optimal vector quantizer design by stochastic relaxation. *IEEE Transactions on Signal Processing*, 40(2):310–322, 1992. → pages 48, 52, 55, 57, 58, 76

T. Zhang and J. Wang. Collaborative quantization for cross-modal similarity search. In *Computer Vision and Pattern Recognition*, pages 2036–2045, 2016. → pages 13

T. Zhang, C. Du, and J. Wang. Composite quantization for approximate nearest neighbor search. In *International Conference on Machine Learning*, pages 838–846, 2014. → pages 3, 4, 13, 18, 20, 30, 35, 37, 38, 39, 40, 41, 44, 50, 60, 64, 69, 71, 91, 96

T. Zhang, G.-J. Qi, J. Tang, and J. Wang. Sparse composite quantization. In *Computer Vision and Pattern Recognition*, pages 4548–4556, 2015. → pages 13, 35, 36, 37, 38, 39, 40, 45, 71