TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Ezgi Can Ozan

Vector Quantization Techniques for Approximate Nearest Neighbor Search on Large-Scale Datasets

Julkaisu 1501 • Publication 1501

Tampere 2017

Tampereen teknillinen yliopisto. Julkaisu 1501
Tampere University of Technology. Publication 1501

Ezgi Can Ozan

Vector Quantization Techniques for Approximate Nearest Neighbor Search on Large-Scale Datasets

Thesis for the degree of Doctor of Philosophy to be presented with due permission for public examination and criticism in Sähkötalo Building, Auditorium SA203, at Tampere University of Technology, on the 13th of October 2017, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2017

Doctoral candidate: Ezgi Can Ozan
Signal Processing
Faculty of Computing and Electrical Engineering
Tampere University of Technology
Finland

Supervisor: Moncef Gabbouj, Professor
Signal Processing
Faculty of Computing and Electrical Engineering
Tampere University of Technology
Finland

Pre-examiners: Niculae Sebe, Professor
Information Engineering and Computer Science
University of Trento
Italy

Wu-Jun Li, Associate Professor
Department of Computer Science and Technology Nanjing University
China

Opponent: Petri Myllymäki, Professor
Department of Computer Science
University of Helsinki
Finland

ISBN 978-952-15-4012-7 (printed)
ISBN 978-952-15-4027-1 (PDF)
ISSN 1459-2045

Abstract

The technological developments of the last twenty years are leading the world to a new era. The invention of the internet, mobile phones and smart devices are resulting in an exponential increase in data. As the data is growing every day, finding similar patterns or matching samples to a query is no longer a simple task because of its computational costs and storage limitations. Special signal processing techniques are required in order to handle the growth in data, as simply adding more and more computers cannot keep up.

Nearest neighbor search, or similarity search, proximity search or near item search is the problem of finding an item that is nearest or most similar to a query according to a distance or similarity measure. When the reference set is very large, or the distance or similarity calculation is complex, performing the nearest neighbor search can be computationally demanding. Considering today's ever-growing datasets, where the cardinality of samples also keep increasing, a growing interest towards approximate methods has emerged in the research community.

Vector Quantization for Approximate Nearest Neighbor Search (VQ for ANN) has proven to be one of the most efficient and successful methods targeting the aforementioned problem. It proposes to compress vectors into binary strings and approximate the distances between vectors using look-up tables. With this approach, the approximation of distances is very fast, while the storage space requirement of the dataset is minimized thanks to the extreme compression levels. The distance approximation performance of VQ for ANN has been shown to be sufficiently well for retrieval and classification tasks demonstrating that VQ for ANN techniques can be a good replacement for exact distance calculation methods.

This thesis contributes to VQ for ANN literature by proposing five advanced techniques, which aim to provide fast and efficient approximate nearest neighbor search on very large-scale datasets. The proposed methods can be divided into two groups. The first group consists of two techniques, which propose to introduce subspace clustering to VQ for ANN. These methods are shown to give the state-of-the-art performance according to tests on prevalent large-scale benchmarks. The second group consists of three methods, which propose improvements on residual vector quantization. These methods are also shown to outperform their predecessors. Apart from these, a sixth contribution in this thesis is a demonstration of VQ for ANN in an application of image classification on large-scale datasets. It is shown that a k-NN classifier based on VQ for ANN performs on par with the k-NN classifiers, but requires much less storage space and computations.

Preface

This study has been carried out at MUVIS group of Tampere University of Technology (TUT), Finland during the years 2012-2017.

First, I would like to express my gratitude to my supervisor Professor Moncef Gabbouj and Professor Serkan Kiranyaz for their support and guidance. I would also like to thank other co-authors and all members of the MUVIS group.

I would like to dedicate this thesis to my wife Gülcan. None of this would be possible without her.

Tampere 1.9.2017

Ezgi Can Ozan

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

Latin alphabet:

a_k: k^{th} affine shift vector.

$\boldsymbol{b}$: a binary selection vector.

$\boldsymbol{B}$: a binary selection matrix.

$\boldsymbol{c}$: a codevector.

$\boldsymbol{C}$: a codebook matrix.

$dim(\boldsymbol{C}^{(l)})$: the number of centroids on the l^{th} dimension of codebook $\boldsymbol{C}$.

D: the number of dimensions.

$\boldsymbol{D}$: a block diagonal matrix.

$e(m, l)$: the graph connection indicator.

$E[\boldsymbol{x}]$: the expected value of random variable $\boldsymbol{x}$.

H: the number of codevector candidates.

K: the number of codevectors.

$len(\boldsymbol{y})$: the length of a binary string $\boldsymbol{y}$.

l_2: the Euclidean norm.

L: the number of dimensions after dimension reduction.

$\boldsymbol{L}$: a label set.

M: the number of quantization layers.

N: the number of samples.

$\boldsymbol{p}$: a nearest neighbor vector.

$\boldsymbol{p}^{(m)}$: the vector $\boldsymbol{p}$ projected to the m^{th} orthogonal subspace.

$\boldsymbol{q}$: a query vector.

$Q(\boldsymbol{x})$: a quantizer.

$\boldsymbol{r}_m$: the residual vector at the m^{th} layer.

$\boldsymbol{R}$: the rotation matrix.

$\boldsymbol{R}^{\perp}$: the null space of $\boldsymbol{R}$.

$\boldsymbol{S}_{i,j}$: a similarity value between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$.

$\boldsymbol{w}_k$: the k^{th} projection vector.

$\boldsymbol{W}$: the principal component matrix.

$\boldsymbol{x}$: a sample vector.

$\widehat{\boldsymbol{x}}$: a quantized vector.

$\boldsymbol{X}$: a set of sample vectors.

$\overline{\boldsymbol{X}}$: ordered reference set

$\boldsymbol{y}$: a binary string.

Greek alphabet:

$\alpha(\boldsymbol{x})$: an encoder function.

$\beta(\boldsymbol{x})$: a decoder function.

∇: the gradient operation.

ϵ: a small constant.

γ: the learning rate.

$\varkappa$: the number of subquantizers for LOPQ.

λ: a penalty parameter.

$\boldsymbol{\mu}$: an affine shift vector.

σ: a standard deviation.

Symbols and scripts:

$\mathcal{B}$: the number of bits.

$\mathbb{C}$: a classifier.

$d(\boldsymbol{x}, \widehat{\boldsymbol{x}})$: the distortion measure between $\boldsymbol{x}$ and $\widehat{\boldsymbol{x}}$.

$\boldsymbol{\mathcal{F}}$: an affine subspace.

$\mathcal{H}_{\mathcal{b}}$: hash function for the $\mathcal{b}^{th}$ bit

$\mathcal{K}$: the number of subspaces for MPCA-E and KSSQ.

$\boldsymbol{\mathcal{M}}$: a distance metric.

$\mathcal{N}_{\boldsymbol{w}_k}$: the neighbors of the winner neuron.

$r\big(Q(\boldsymbol{x})\big)$: the instantaneous rate of a quantizer Q.

Abbreviations:

ANN: Approximate Nearest Neighbor Search

APQ: Additive Product Quantization

AQ: Additive Quantization

CKM: Cartesian K-Means

CompQ: Competitive Quantization

CQ: Composite Quantization

ERVQ: Optimized (Enhanced) Residual Vector Quantization

FLANN: Fast Library for Approximate Nearest Neighbors

ICM: Iterative Conditional Modes

ILP: Integer Linear Programming

IMI: Inverted Multi-Index

ITQ: Iterative Quantization

IVFADC: Inverted File with Asymmetric Distance Computation

JKM: Joint K-Means Quantization

KSSQ: K-Subspaces Quantization

LOPQ: Locally Optimized Product Quantization

LSH: Locality Sensitive Hashing

LUT: Look-up table

MPCA-E: M-PCA Binary Embedding

MSE: Mean-squared error

NN: Nearest Neighbor Search

OCKM: Optimized Cartesian K-Means

OPQ: Optimized Product Quantization

OTQ: (Optimized) Tree Quantization

PCA: Principal Component Analysis

PCA-E: PCA-Embedding

PCM: Pulse Coded Modulation

PQ: Product Quantization

RVQ: Residual Vector Quantization

SH: Spectral Hashing

SOBE: Self-Organized Binary Encoding

SOM: Self-Organizing Maps

SVM: Support Vector Machine

TC: Transform Coding

VQ: Vector Quantization

List of Publications

[P1] E. C. Ozan, S. Kiranyaz and M. Gabbouj, "M-PCA Binary Embedding for Approximate Nearest Neighbor Search," *2015 IEEE Trustcom/BigDataSE/ISPA*, Helsinki, 2015, pp. 1-5.

[P2] E. C. Ozan, S. Kiranyaz and M. Gabbouj, "K-Subspaces Quantization for Approximate Nearest Neighbor Search," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 7, pp. 1722-1733, July 1 2016.

[P3] E. C. Ozan, S. Kiranyaz, M. Gabbouj and X. Hu, "Self-organizing binary encoding for Approximate Nearest Neighbor search," *2016 24th European Signal Processing Conference (EUSIPCO)*, Budapest, 2016, pp. 1103-1107.

[P4] E. C. Ozan, S. Kiranyaz and M. Gabbouj, "Joint K-Means quantization for Approximate Nearest Neighbor Search," *2016 23rd International Conference on Pattern Recognition (ICPR)*, Cancun, Mexico, 2016, pp. 3645-3649.

[P5] E. C. Ozan, S. Kiranyaz and M. Gabbouj, "Competitive Quantization for Approximate Nearest Neighbor Search," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 11, pp. 2884-2894, Nov. 1 2016.

[P6] E. C. Ozan, E. Riabchenko, S. Kiranyaz and M. Gabbouj, "A vector quantization based k-NN approach for large-scale image classification," *2016 Sixth International Conference on Image Processing Theory, Tools and Applications (IPTA)*, Oulu, 2016, pp. 1-6.

1 Introduction

With the arrival of the digital age, data became a part of our daily lives. In the beginning we were on the consuming side, we listened to music on the radio, watched movies on television, talked to our friends through cell phones while walking on the road and browsed the internet to find the nearest restaurant. Then we started to contribute to data ourselves, by writing comments about our favorite restaurants, joining social networks to catch up with old pals and uploading videos to the internet. As the technology advanced, more and more people gained access to data and it became easier to both consume and contribute. This is how “Big Data” was born.

Although data analytics has been a research topic for almost a hundred years, with the introduction of the term Big Data to our lives, it gained a new perspective, which also caused data science to attract great attention. Of course, century old techniques were not sufficient to extract the required information from the Big Data. They were either too slow to handle such amount or had storage space issues. Hence, demand towards new approaches, which are suitable for Big Data emerged. Especially, as the semiconductor technology started to reach the limits of its power, providing more and faster servers alone could not keep up with this growth. Therefore, new methods needed to be developed specifically for Big Data, methods that can scale up to billions of samples and run in parallel in thousands of servers. Some came as updates to traditional techniques while some are created from scratch.

With the exponential increase in the digital data, providing efficient similarity search has gained greater importance. Besides the growth in the amount of data, with the improvements in pattern recognition and machine learning, the descriptors also began to grow. The learned descriptors obtained from state-of-the-art deep learning algorithms generated much bigger feature vectors than engineered features. These affected the similarity search in three ways: First, there were more samples to compare. Second, each comparison demanded more computational power. The third and the last, the number of

samples, which could be compared and ordered at the same time, was limited by the memory. Less samples could be loaded to RAM at once as the dimension of the samples increased.

In this thesis, five novel methods, which aim to solve the problems aforementioned above, are presented. The proposed methods perform fast and efficient similarity search on large-scale datasets. Dataset compression is provided in extreme levels, reaching up to hundreds. Distance calculation between samples is also accelerated; the computational cost is reduced to a few look-ups from a pre-calculated table, whose size is negligible compared to the size of the compressed set. The detailed tests performed on distinguished benchmarks show that the proposed methods outperform the prior work, leading to the state-of-the-art results in this field. The application areas of the proposed methods varies from query by example to instance based classification or regression. An example application is presented as the sixth contribution, presenting the advantage of choosing the selected approach on large-scale datasets.

1.1 Objectives and Outline of the Thesis

This thesis aims to provide novel methods with improved performance and efficiency in the field of Vector Quantization for Approximate Nearest Neighbor Search. The objectives of this thesis can be listed as given below:

- to give a detailed description of Approximate Nearest Neighbor Search problem and explain why it is so important.
- to investigate comprehensively the solutions proposed for this problem in the literature.
- to propose novel methods in order to provide efficient and high performance solutions.

The thesis is organized as follows: First, Vector Quantization problem is described in Chapter 2, where also a brief historical review about quantization and vector quantization is provided. In Chapter 3, the Approximate Nearest Neighbor Search problem is defined, and several solution techniques from the literature are briefly investigated. In Chapter 4 the contributions of this thesis to the literature are presented in detail and finally, Chapter 5 concludes the thesis.

1.2 Publications and Author's Contribution

In [P1], subspace clustering techniques are introduced to the vector quantization for approximate nearest neighbor search problem and the quantization performance is shown to improve. The author implemented the proposed method, performed the tests and wrote the paper, together with the supervisors' fruitful discussions and reviews.

In [P2], the computational requirements of [P1] are improved and the idea that suggests performing quantization in multiple subspaces is shown to be applicable. The proposed scheme gives notably better results than the prior work. The author implemented the proposed method, performed the tests and wrote the paper, together with the supervisors' fruitful discussions and reviews.

In [P3], Self-Organizing Maps, which is a popular data visualization and clustering algorithm, is introduced to the approximate nearest neighbor search problem. The author implemented the proposed method, performed the tests and wrote the paper, together with the supervisors' fruitful discussions and reviews.

In [P4], a hierarchical structure based method is proposed to improve the quality of codebook generation for quantization, in order to achieve better nearest neighbor approximations. The author implemented the proposed method, performed the tests and wrote the paper, together with the supervisors' fruitful discussions and reviews.

In [P5], a detailed analysis of the vector quantization problem and the distance approximation approach based on vector quantization is provided. In addition, a solution to this problem based on stochastic gradient descent optimization, which is a well-proven optimization method is proposed. The author implemented the proposed method, performed the tests and wrote the paper, together with the supervisors' fruitful discussions and reviews.

In [P6], a demonstration of the applicability of the aforementioned solutions to classification problem is demonstrated by introducing a vector quantization based k-Nearest Neighbor classifier. The author implemented the proposed method, performed the tests together with Ekaterina Riabchenko and wrote the paper, together with the supervisors' fruitful discussions and reviews.

2 Vector Quantization

Quantization can be defined as division of a continuous value into smaller discrete values [1]. Usually, signals are continuous, which means that there are infinitely many possibilities for the value of a signal. When it is not convenient or efficient to deal with so many options, signals are digitized and the continuous signals are mapped to a smaller set of predefined values. Quantization is the process of determination of these values. Hence, quantization plays a very important role in building the bridge between the analogous real life and its digital representation. Quantization finds applications in a wide range of fields such as electronics [2], optics [3], biology [4], physics [5], chemistry [6], etc...

In this chapter, first, an introduction to the quantization problem is presented. Mathematical definition of the problem is provided, and important concepts of quantization are further investigated. Then a summarized review is given, presenting the evolution of quantization methods throughout the history.

2.1 Quantization: Rate, Distortion and Optimality

In the most general sense, the definition of quantization can be formulated as follows: A quantizer Q transforms a D-dimensional sample $\boldsymbol{x} \in \mathbb{R}^D$ to its corresponding quantized value $\hat{\boldsymbol{x}} \in \mathbb{R}^D$. If $D = 1$, the quantizer is called *scalar*, else it is a *vector quantizer*. The quantizer is formed of three structures: an encoder, a decoder and a codebook. An encoder $\alpha: \boldsymbol{x} \rightarrow \boldsymbol{y}$, maps the sample $\boldsymbol{x}$ to its corresponding code $\boldsymbol{y}$, which is usually a string of bits. A decoder $\beta: \boldsymbol{y} \rightarrow \hat{\boldsymbol{x}}$, maps the code $\boldsymbol{y}$ to $\hat{\boldsymbol{x}}$, which is the reconstructed version of $\boldsymbol{x}$. So, a quantizer Q is defined as

$$Q(\boldsymbol{x}) = \beta\big(\alpha(\boldsymbol{x})\big) \tag{2.1}$$

The *rate* of a quantizer is the length of the code, i.e., the number of bits that is required for quantization of a sample. The instantaneous rate of a quantizer for a given input $\boldsymbol{x}$ can be defined as follows:

$$r\big(Q(\boldsymbol{x})\big) = \frac{1}{D} len\big(\alpha(\boldsymbol{x})\big) \tag{2.2}$$

Distortion is a measure, which is defined to measure the quality of reconstruction, i.e., the similarity of the reconstructed sample $\hat{\boldsymbol{x}}$ to the actual sample $\boldsymbol{x}$. An ideal distortion measure is non-negative, easy to compute and perceptually meaningful. There are different distortion measure proposed in the literature [1], but the most frequently used one is the l_2-Norm, and it can be defined as follows:

$$d(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2 = (\boldsymbol{x} - \hat{\boldsymbol{x}})^T(\boldsymbol{x} - \hat{\boldsymbol{x}}) \tag{2.3}$$

Therefore, the performance of the quantizer can be measured by these two values: average rate and distortion. Both values are defined as follows:

$$\begin{aligned} Rate&: E[r(\boldsymbol{x})] = \frac{1}{D} E\big[len\big(\alpha(\boldsymbol{x})\big)\big] \\ Distortion&: E[d(\boldsymbol{x}, \hat{\boldsymbol{x}})] = E[\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2] \end{aligned} \tag{2.4}$$

It is often aimed to find an optimum point for the rate-distortion trade-off. The optimization can be performed in various ways, for example, constraining one and optimizing the other. However, for the fixed-rate problems[1], the optimization problem reduces to minimization of distortion only. The *optimality conditions* for a fixed-rate, fixed-dimension quantizer is first defined by Lloyd [7]. These conditions can be summarized as follows:

- Given an encoder α, and a set of samples $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\}$, the optimal decoder β is given by the minimization of the following equation:

$$\beta(i) = \underset{\boldsymbol{z}}{\operatorname{argmin}}\, E[d(\boldsymbol{X}, \boldsymbol{z}) | \alpha(\boldsymbol{X}) = i] \tag{2.5}$$

As shown above, the optimality condition is met when the conditional expectation of the distortion is minimized, given the encoder output is $\alpha(\boldsymbol{X}) = i$. $\beta(i)$ which satisfies the above minimization is a Lloyd centroid [7]. Here note that, the optimal decoder output is

[1] In this thesis, the focus is set on a special type of quantizers, where the rate is constant. In other words, the number of bits that is used for quantization of each sample is the same. These sort of quantizers are called *fixed-rate* or *fixed-length* quantizers.

also the optimal estimate of the given sample. The optimal estimate is given by the following equation:

$$\beta(i) = E[\boldsymbol{X}|\alpha(\boldsymbol{X}) = i] \tag{2.6}$$

- Given a decoder β, and a sample $\boldsymbol{x}$, the optimal encoder α is given by the minimization of the following equation:

$$\alpha(\boldsymbol{x}) = \underset{i}{\operatorname{argmin}}\, d\big(\boldsymbol{x}, \beta(i)\big) \tag{2.7}$$

According to the statement given above, for a given input, the optimality condition is met when the code, which provides the minimum distortion, i.e., its corresponding nearest neighbor, is selected.

Another important aspect of quantizers is complexity. Complexity can be investigated in two ways: First is the computational complexity, which is related to the total number of performed arithmetic operations. The second is the storage complexity, which is the amount of additional space that is required[2]. Therefore, a quantizer can be defined and evaluated in terms of three features: average rate, average distortion and complexity in computation and storage.

2.2 A Historical Review

The first examples of quantization analysis go back to 19th century. Sheppard analyzed rounding off the real numbers to their nearest integers in estimation of histogram densities, proposing the first uniform quantizer [8]. A quantizer is *uniform* if it has equal quantization levels and the thresholds are between the adjacent levels. In 1938, Pulse Coded Modulation (PCM), which is the first digital technique for conveying an analogue signal through an analogue channel, is proposed [9]. The first PCM contained a sampler, a

[2] There is often a trade-off between computational and storage complexity measures, for example, arithmetic operations can sometimes be evaded using additional look-up tables, which requires additional storage space.

quantizer and a binary pulse modulator[3]. Quantization of sampled signals provides an approximation. Further analysis of PCMs set the stones for first attempts of theoretical analysis of quantization [10]. The *6-dB rule*, which states that the signal-to-noise ratio of a uniform quantizer increases 6-dB for each one bit increase, is discovered by Bennet during analysis of PCM for communication channels [11]. Bennet also showed that, under certain assumptions, the quantization error could be modeled as an additive white noise. This later led to quantization with addition of a dither signal, which is used to improve the quality of quantized images [12]. A very important study on quantization is published by Lloyd in 1982 [7]. In this publication, Lloyd defined the *optimality conditions* for a fixed-rate quantizer. Then using this optimality, Lloyd proposed an iterative quantizer using a minimum error mapping. This led to Lloyd's quantizer, which is similar to today's popular K-Means algorithm [13].

When it is discovered that scalar quantization fails to exploit the redundancies, which occur in quantization of correlated sources such as images and speech, it was proposed to combine linear signal processing techniques with quantization, which lead to *predictive coding* and *transform coding* [1]. Predictive quantizers have memory so that quantization of a sample depends on the previous samples [14]. Predictive coding has been used in many coding standards including MPEG and H.26X series [1]. Transform coding creates a vector form the samples and multiplies this vector with an orthogonal transform. The resulting transforms are quantized separately [15]. Transform coding is used extensively for image and video coding together with the discrete cosine transform [16].

When Shannon published his work about lossless coding theory in 1948 [17], he showed that when equal number of bits are assigned for all quantization cells, it is not optimal unless the cells have equal probabilities. This lead to variable-rate quantization. Huffman proposed an algorithm [18], which provides the smallest binary codes for a set of given probabilities. Today this is known as Huffman coding, which is used in current JPEG and related standards [19]. One drawback of this technique is that the produced codes have variable bit length and handling of this requires additional operations.

Earlier studies focused on scalar quantization as expected, and its generalization to multi-dimensions was rather a theoretical discussion. Until 60s, the quantization problem was mostly related with PCM, or analog-digital conversion. Vector Quantization (VQ) acted as a model when the limits of quantization were discussed. VQ became an actual

[3] Later PCMs converged to a point where binary pulse modulator is replaced with an encoding module.

problem when naive applications of scalar quantization to vectors became infeasible, as the number of dimensions started to increase. Before, permutations of scalar quantizer coefficients were reconstructed as codevectors, and a vector was quantized by applying brute-force nearest neighbor search among all the codevectors. This method is called permutation coding [20]. The computational and storage requirements of permutation coding motivated the researchers to seek for solutions that are more efficient [1]. The first practical VQ methods were targeting the permutation coding directly and aiming to improve the complexities requirements [21], [22]. One of the earliest studies, which targets the vector quantization as a coding problem belonged to Steinhaus [23]. In this work, the scalar quantization problem was generalized to 3-dimensional space and this space was partitioned into disjoint cells. Alternatively, clustering algorithms were introduced to VQ problem [24]. With these algorithms, came the first examples of VQ in image and audio processing [25], [26].

Later, structured designs were proposed in order to improve the coding efficiency. Lattice quantization is a structured quantization method, where the reconstructed codevectors are constrained to be from a lattice [27]. The final partitioning results in cells having the same shape and orientation. Another structured method is product quantization, which uses codevectors that are reconstructed as Cartesian products of lower dimensional codevectors [28]. With separation of codebooks, obtaining lower computational complexity is aimed. Product quantization has variants such as shape-gain quantizer [29] and pyramid quantizer [30]. Transform coding is a special case of product quantization, where the quantized vectors are transformed using an orthogonal transformation and sub-vectors are scalars, i.e., the lower dimension is 1 [31]. Similarly, subband coding and wavelet coding are also related to transform coding [32]. Subband codes perform decomposition on an image by using linear filters to obtain the subband images. Wavelet codes can be interpreted as subband codes with logarithmically varying subbands. Tree-Structured VQ [33] generates a binary tree structure, where each node contains a codevector. The quantized vector propagates in the tree by searching for the closer codevector. Multistage vector quantization [34] is a form of tree-structured VQ, which uses a single codebook for all the branches, and the residual error is propagated to the next level.

So far, the pioneering algorithms have been addressed to provide an insight to the historical evolution of quantization. Recent approaches mostly focus on application-based optimizations of previous methods or derivation of new variants of old techniques [35]–[41]. The reader is referred to two comprehensive surveys for further reading about quantization and vector quantization [1], [42].

In this thesis, VQ for Approximate Nearest Neighbor Search (ANN), a recent approach, which consists of VQ algorithms that are developed in order to solve the ANN problem for very large datasets, is investigated. Chapter 3 presents in details the ANN problem and the main challenges encountered when solving this problem especially for large-scale data sets, and explores how VQ can be exploited in this direction.

3 Approximate Nearest Neighbor Search

In this chapter, an introduction to the Approximate Nearest Neighbor Search (ANN) problem is provided. First, the Nearest Neighbor Search (NN) problem is defined mathematically. Then the reasons of proposing approximate solutions for this problem are presented. Important approaches for ANN are grouped into three subchapters as Partition Trees, Hashing and Vector Quantization, the latter being the main focus of the thesis.

3.1 Nearest Neighbor Search

The Nearest Neighbor Search (NN) problem is to find a point, which is "nearest" to a given query, among a given set of points [43]. The "nearness" of two points is calculated by a given distance metric [44]. NN is a significant problem in many research areas including but not limited to computational geometry [45], computer vision [46], [47], data mining, machine learning and pattern recognition [48]–[52]. NN forms the root of many retrieval applications in various fields such as image ranking and retrieval [53], audio information retrieval [54], multimedia information retrieval [55]. Moreover in machine learning, the instance-based learning is derived from the nearest neighbor classifier, which relies on NN [56].

The NN problem can be formulated as follows: given a set of samples $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\}$, ($\boldsymbol{x} \in \mathbb{R}^D$) and a query $\boldsymbol{q} \in \mathbb{R}^D$, find the sample $\boldsymbol{p}$ that is the nearest to the given query $\boldsymbol{q}$, for a given metric $\boldsymbol{\mathcal{M}}: D \times D \rightarrow \mathbb{R}$

$$\boldsymbol{p} = \underset{\boldsymbol{x} \in \boldsymbol{X}}{\operatorname{argmin}} \boldsymbol{\mathcal{M}}(\boldsymbol{q}, \boldsymbol{x}). \tag{3.1}$$

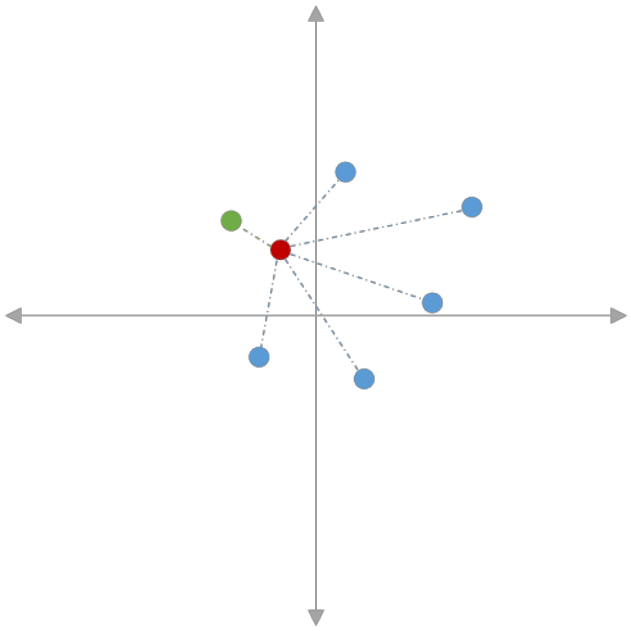

Figure 1. A 2-dimensional Nearest Neighbor Search Illustration. For the given red query point, the green point is the nearest neighbor.

This problem has a naïve solution, which requires the calculation of $\mathcal{M}(\boldsymbol{q}, \boldsymbol{x})$ for all $\boldsymbol{x} \in \boldsymbol{X}$, resulting in a query time which is linear in N and D ($O(DN)$) [43]. Data structures have been proposed to reduce the query time to sublinear complexity, with some preprocessing overhead. One of the first approaches to decrease this complexity was proposed by Dobkin and Lipton [57], providing a query time of complexity $O(2^D \log N)$ and a preprocessing cost of $O(2^{D+1})$. Similarly, following proposals [58], [59] all ended up with a complexity that is exponential (or polynomial) with D. This situation, which can be phrased as the "curse of dimensionality" [43], shows that exact nearest neighbor solutions are impractical for $D > 2$ [43], [52], [60].

3.2 Approximate Nearest Neighbor Search

With the growth of datasets and descriptor cardinalities, the need for NN algorithms with significantly smaller query time requirements emerged. This has led to the emergence of approximate solutions for nearest neighbor search [43]. ANN can be formulated as follows: given a set of samples $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\}$, (sample $\boldsymbol{x} \in \mathbb{R}^D$ is a D-dimensional

vector) and a query $\boldsymbol{q} \in \mathbb{R}^D$, find a sample set $\boldsymbol{P} \subseteq \boldsymbol{X}$ that is in a neighborhood of the given query $\boldsymbol{q}$, for a given metric $\boldsymbol{\mathcal{M}}: D \times D \rightarrow \mathbb{R}$

$$for\ all\ \boldsymbol{p} \in \boldsymbol{P}\ and\ \boldsymbol{x} \in \boldsymbol{X}, \qquad \boldsymbol{\mathcal{M}}(\boldsymbol{q}, \boldsymbol{p}) \leq (1 + \varepsilon)\boldsymbol{\mathcal{M}}(\boldsymbol{q}, \boldsymbol{x}). \tag{3.2}$$

ANN relaxes the constraint in NN, which requires the resulting sample $\boldsymbol{p}$ to be the exact nearest to the given query $\boldsymbol{q}$. In practice, when a wide range of applications for NN is considered, the exact NN can easily be replaced by ANN, as the comprehensive body of literature in ANN also suggests [61]–[65].

The simple formulation in (3.2) provides a generic mathematical definition for error-bounded ANN [66]. ANN can also be defined as time-bounded if the time spent during the search is limited [67]. These definitions can be extended or modified with additional constraints depending on the use cases. In this chapter, we group ANN algorithms proposed in the literature into three major categories: Partition Tree Structures for ANN, Hashing for ANN and Vector Quantization for ANN. In the following subchapters, these categories are discussed in detail.

3.2.1 Partition Tree Structures for ANN

Partition trees are data structures, which aim to divide the dataset hierarchically in order to organize the samples. Each node of the tree corresponds to a subpartition of the dataset, forming a hierarchical organization. From the root node, which contains the whole dataset, each traverse downwards lead to a disjoint subset. The leaf nodes may correspond to either a single sample or a set of samples.

The methods discussed in this category are mainly derived from K-d Trees [68], which is one of the earliest and most popular partition tree structures. Proposed in mid 70's, this method aims to extend the one-dimensional binary search to multi-dimensions. D-*dimensional* vectors are stored in a K-d tree structure, in which every branch splits the dataset into two, by using the median value of a selected dimension as a division threshold [69]. The sample vectors are stored as leaves in the constructed tree. An illustration of a 2-dimensional K-d tree is presented in Figure 2.

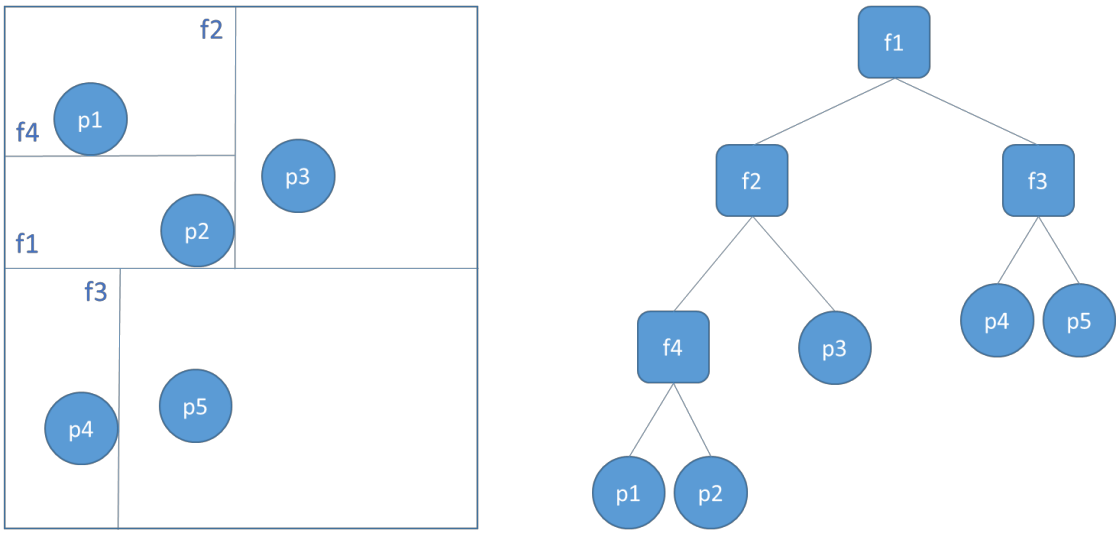

Figure 2. A 2-dimensional K-d Tree Structure Illustration. On the left, the dataset points are shown together with the lines separating the branches. On the right, the corresponding three structure is shown.

Given a novel query, the values in the selected dimensions are compared to the corresponding threshold values and the tree is traversed according to the selected subbranch at each node. The binary split is performed at each level; hence, the maximum depth of the tree is bounded by $\log_2 N$. As each split is decided by a selected dimension, it may not be efficient to use K-d Trees on datasets where $D > \log_2 N$, as there will be discarded dimensions. A rule of thumb is given in [68], K-d Trees might be used efficiently only if $N > 2^{2D}$.

As the first candidate obtained may not necessarily be the nearest neighbor, it is usually followed by further searches, which look for better candidates, referred as *"priority search"* or *"backtracking"* [66], [67], [69], [70]. It creates a priority queue using the sub-trees, in order to access the data samples with greater probabilities being the true nearest neighbor are accessed first. Several other methods are derived from K-d Trees. PCA-Trees method [71] aims to use the direction of principal components, which are obtained from Principal Component Analysis (PCA) [72]. RP-Trees [73] and Spill-Trees [74] generate a set of random projections and use the best ones to form the partition hyperplanes. RP-Trees, Spill-Trees and PCA-Trees all aim to propose better hyperplanes for partitioning. Selecting among a set of projections or using the principal component directions lead to better representation of the training set. However, an inner-product operation is required

at each node, which has a complexity of $O(D)$ while in K-d Trees the complexity of selecting branches is $O(1)$. Hierarchical K-Means Trees [75] proposes using Voronoi tessellations to partition each layer. This approach relaxes the binary separation constraint to obtain better separation performance, but at the expense of increasing the number of operations required for branching. The complexity of each branching operation is $O(KD)$, where K is the number of centroids. Multiple randomized K-d Trees [76] divides the space using multiple trees, which results in more partitions. When a novel sample is queried, the query is performed simultaneously in multiple trees, leading to a significant performance improvement. FLANN [77] combines multiple randomized K-d Trees with hierarchical K-Means and automatically finds the best parameters for the given dataset. Trinary Projection Trees [78] uses a linear combination of coordinate axes weighted by 1, 0 and -1 in order to form the projection hyperplanes. They use the maximum variance criterion to determine the trinary-projection directions.

As mentioned above, Partition Trees are data structures implemented on a dataset in order to organize samples and provide fast access to a desired set of points. Because of their organization schemes, Partition Trees usually keep the original dataset, especially if backtracking or priority search is used. They also require additional space to store the partition hyperplane parameters, which can sometimes be comparable to the size of the dataset itself, depending on the depth of the tree and used hyperplane functions [78]. While non-linear query times are observed for lower dimensional cases, as dimensions grow, the query times may not be much less than the linear search [79]. According to these properties of Partition Trees, it can be said that alternative approaches are required for Big Data, which especially require much less storage space.

3.2.2 Hashing for ANN

Hashing is another branch of proposed solutions for ANN and it is well investigated in the literature. Generally speaking, hashing can be described as a transformation or a mapping between the dataset elements and binary codes [63]. Samples in a dataset are encoded into binary strings using functions called hash functions. Unlike the traditional hashing technique in computer science, where it is not desirable for two samples to share the same hash code, here similar samples are aimed to have similar hash codes.

Given a sample $\boldsymbol{x} \in \mathbb{R}^D$, a $\mathcal{B}$-bit binary code $\boldsymbol{y} = \{\boldsymbol{y}[1], \boldsymbol{y}[2], \dots, \boldsymbol{y}[\mathcal{B}]\}$ can be obtained using $\mathcal{B}$ hash functions as $\boldsymbol{y} = \{\mathcal{H}_1(\boldsymbol{x}), \mathcal{H}_2(\boldsymbol{x}), \dots, \mathcal{H}_{\mathcal{B}}(\boldsymbol{x})\}$. So the hash function $\mathcal{H}_{\ell}$ is a mapping from $\mathbb{R}^D \rightarrow \mathbb{B}^{\mathcal{B}}$. Once the binary codes are calculated, ANN search is performed

using the Hamming distance[4]. The advantage of the Hamming distance is that, it can be implemented using very fast bitwise logic operations (XOR followed by pop-count [80]). This improves the search speed significantly. The approximation of the original distance by Hamming distance can be formulized as follows:

$$\begin{aligned} \boldsymbol{y_q} &= \mathcal{H}(q) \\ \boldsymbol{y_p} &= \mathcal{H}(p) \end{aligned} \tag{3.3}$$

$$\boldsymbol{\mathcal{M}(q,p)} \sim \left\| \boldsymbol{y_q} - \boldsymbol{y_p} \right\|_H$$

Locality Sensitive Hashing (LSH) is initially proposed in [43] and a significant number of hashing algorithms are derived from it. LSH proposes to find a family of hash functions, which maps the similar samples to the same hash codes with a higher probability than dissimilar samples. In other words, a family of hash functions $\mathcal{H}$ has locality-sensitiveness for a radius R, a constant $a > 1$ and two probabilities P_1 and P_2 $(P_1 > P_2)$ for two given samples p and q,

$$\begin{aligned} if\ dist(p,q) \leq R, \quad & \text{then} \quad Prob[\mathcal{H}(p) = \mathcal{H}(q)] \geq P_1 \\ if\ dist(p,q) \geq aR, \quad & \text{then} \quad Prob[\mathcal{H}(p) = \mathcal{H}(q)] \leq P_2 \end{aligned} \tag{3.4}$$

In an LSH scheme, with such a hash function family as described above, all data samples are encoded according to the outputs of the hash functions. Each item is placed into its corresponding *"hash bucket"*. In other words, for a given query q, all the dataset samples placed in the bucket $\mathcal{H}(q)$ are considered as the nearest samples to q [43]. An illustration of LSH is given in Figure 3.

[4] Being the most frequently used metric, the Hamming distance is not the only alternative for the hashing technique. There are other binary distances proposed for specific hashing algorithms [63].

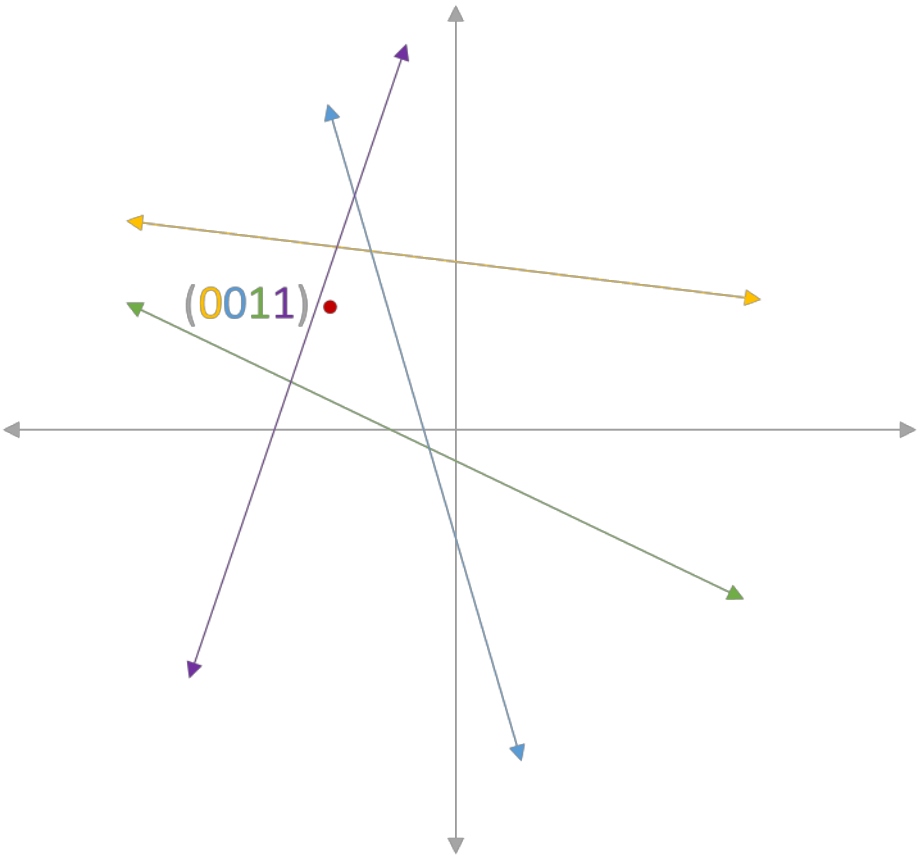

Figure 3. A 2-dimensional LSH Illustration. Colored arrows represent random hash functions. Red dot is the encoded sample point.

LSH and its variants use randomly generated linear projections as hash functions. The method used to generate these projections can vary, or there can be slight modifications on the final hash function [63]. Nevertheless, the hash functions in general can be defined as follows:

$$\mathcal{H}_k(\boldsymbol{x}) = sgn\left(\boldsymbol{w}_k^T \boldsymbol{x} + a_k\right) \tag{3.5}$$

where, $\boldsymbol{w}_k$ is a D-dimensional projection vector and a_k is an affine shift. Different hash methods are derived from LSH for different distances or similarities. For example, for binary vectors, where a similarity measure can be defined using the Hamming distance, a Binary LSH variant is proposed in [43]. Another LSH technique, which uses p-stable

distributions[5] to generate hash functions for l_p distance metrics, is presented in [81]. Spherical LSH [82], which uses randomly rotated polytopes to generate hash functions, is proposed for vectors on a unit hypersphere of the Euclidean space. Similarly, Angle-Based LSH [83] is proposed for angle-based distance approximation. Also in [84] χ^2-LSH is presented for chi-squared distance metric. A more generic approach is presented in [85], where learning a Mahalanobis metric using semi-supervised information is proposed. The hash functions are then generated according to the learnt similarity.

The aforementioned methods are all based on random generations of hash functions; hence, they are independent from the data itself[6]. However, this approach comes with a major drawback. Because of the randomness of hash functions, too many hash functions need to be generated in order to obtain a high precision. This leads to longer codes and greater storage space requirements. In order to solve this problem, "learning-to-hash" algorithms are proposed [64]. Learning-to-hash algorithms aim to use advanced machine learning techniques to learn the hash function $\boldsymbol{y} = \mathcal{H}(\boldsymbol{x})$, which maps the input $\boldsymbol{x}$ to the corresponding binary code $\boldsymbol{y}$, in a way such that the result of the nearest neighbor search in the code space is an efficient approximation of the true nearest neighbor. While learning this mapping, the requirements given in (3.4) are still valid, yet in order to improve the efficiency and performance, more constraints are defined.

One of the earliest and most significant learning-to-hash approaches is Spectral Hashing (SH) [86]. Spectral Hashing defines a good code as the following: First, it should be easy to compute for a novel input, it should not require too long codes to encode a full dataset and it should map similar items to similar codes. In order to produce efficient codes, it is proposed that each bit should have an equal probability of being 0 or 1. Also the bits should be independent of each other [86]. The code generation problem turns into the following optimization problem:

[5] For example, Gaussian distribution is a p-stable distribution.
[6] Except maybe the method proposed in [85], which includes semi-supervised distance learning. However, note that the learning affects the distance measure, not the generation of hash functions.

$$
\begin{aligned}
minimize\colon & \sum_{i,j} \boldsymbol{S}_{i,j} \left\| \boldsymbol{y}_i - \boldsymbol{y}_j \right\|^2 \\
s.t. \quad & \boldsymbol{y}_i \in \{-1,1\}^{\mathcal{B}} \\
& \sum_{i}^{N} \boldsymbol{y}_i = \boldsymbol{0} \\
& \frac{1}{N} \sum_{i}^{N} \boldsymbol{y}_i \boldsymbol{y}_i{}^T = I
\end{aligned}
\tag{3.6}
$$

where $\boldsymbol{S}_{i,j} = exp\left(-\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|^2 / \epsilon^2\right)$ defines the similarity between two samples in the Euclidean space, and ϵ is a normalization constant. $\boldsymbol{I}$ is the identity matrix. As it can be seen, $\boldsymbol{S}_{i,j}$ grows with the similarity so the minimization requires $\left\|\boldsymbol{y}_i - \boldsymbol{y}_j\right\|^2$ to be small. Therefore, the algorithm keeps the neighbors in the original space also close in the code space. The second constraint brings the equal probability while the final constraint requires the bits to be uncorrelated. This optimization is a balanced graph partitioning problem for a single bit, which is NP hard [87]. Instead, another approximate solution for the given optimization is proposed, which uses the 1D Laplacian eigenfunctions, with the assumption that the input data are uniformly distributed. The resulting encoding is a sinusoidal function partitioning the data along the direction of the principal components. SH has led to many variants in the literature. In [88], Self-Taught Hashing uses the same optimization but solves it using spectral relaxation method similar to Ratio-Cut [89]. Then, it trains Support Vector Machine (SVM) classifiers to learn this encoding scheme and encodes the novel samples using these classifiers. Anchor Graph Hashing [90] improves the cost of building the $N \times N$ similarity matrix $\boldsymbol{S}$ by using an approximate similarity matrix that is obtained by a small set of anchor points.

As mentioned above, SH and its variants try to keep the neighbors in the original space also neighbors in the code space. Another approach in order to improve performance of LSH is to find better projections. Among the first examples comes the Principal Component Hashing [91]. It proposes using principal components instead of random projections in LSH. Principal components are orthogonal to each other; hence, hash functions are generated independently. In [92], it is shown that any random orthogonal projection applied on top of PCA provides better hash functions. Iterative Quantization[7] [93] takes this

[7] Although its name indicates otherwise, Iterative Quantization is a hashing method rather than quantization. It starts with a cost definition similar to quantization error, but then ends

approach one step further and iteratively optimizes the rotation to provide the most efficient encoding scheme. Double-bit Quantization [94] uses the same logic with [93] but distributes two bits per projection instead of 1. Isotropic Hashing [95] also looks for a better rotation after PCA, but instead it aims to equalize the variance at each dimension after the rotation.

Although Hamming distance is the most popular distance used in the coding space, there are other approaches, which propose using alternative distances. Manhattan Hashing [96] proposes replacing Hamming distance with Manhattan distance. In [97], each bit of the encoding is weighted in order to obtain a weighted Hamming distance. Furthermore, a query-adaptive weighted Hamming distance scheme is proposed in [98]. Asymmetric distances for several binary encoding methods are presented in [99]. Many other hashing methods can be found in the literature, with different optimization criteria for different purposes. For example, supervised and semi-supervised training[8] are also introduced in hashing [100]–[105]. There are hashing methods which aim to preserve the triplet relations [106], [107] or even the whole ranking order [108]. Deep Hashing [109] uses deep neural networks to embed images directly into binary codes. For other methods and further detailed information about hashing, the reader is referred to [63]–[65].

As discussed, hashing methods aim to transform a dataset from its original space into a binary code space. By doing this, hashing methods aim to first provide fast and efficient ANN, and second compress the dataset significantly. Calculation of Hamming distance with logical bitwise operations considerably increases the speed, and storage of dataset samples in binary codes instead of original space provides extreme level compression. One significant drawback of using Hamming distance (or similar binary distances) arises from its binary nature. Hamming distance is an integer value between zero and the length of the binary string. For a $\mathcal{B}$-bit binary string, there are only $\mathcal{B} - 1$ different distances. Therefore, when ranking is performed using Hamming distance as a metric, there are too many samples having the exact same distance to the given query. This affects the ranking performance crucially. Alternative approaches are proposed in the literature to overcome this drawback.

---

up with binarized centroids and uses Hamming distance to calculate the neighborhood in the code space.

[8] Supervised or semi-supervised approaches are outside the scope of this thesis, as they require manual labeling for the training set. This thesis focuses on unsupervised methods, which produce generic solutions, and do not need manually annotated datasets.

3.2.3 Vector Quantization for ANN

The last branch of ANN algorithms proposes to use VQ techniques to compress datasets and store them in the form of binary strings, similar to hashing methods. However contrary to hashing, approximation of distances between database samples and the query is performed by look-up tables instead of binary distances. This gives a significant improvement in performance [110].

As mentioned in Chapter 2.2, vector quantization can be induced to an optimization problem, where minimization of mean-squared error (MSE) is the objective. MSE of a vector quantizer can be defined as follows: Given a set of N vectors $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$ ($\boldsymbol{x}_i \in \mathbb{R}^D$), for a vector quantizer Q, the mean squared quantization error MSE_Q is defined as

$$MSE_Q = \frac{1}{N}\sum_{i}^{N} \|\boldsymbol{x}_i - Q(\boldsymbol{x}_i)\|_2^2 \tag{3.7}$$

The quantizer Q quantizes the input $\boldsymbol{x}_i$ to its corresponding codevector $\hat{\boldsymbol{x}}_i$ as

$$Q(\boldsymbol{x}_i) = \boldsymbol{C}\boldsymbol{b}_i \tag{3.8}$$

where, $\boldsymbol{b}_i$ is the binary selection vector, and $\boldsymbol{C} \in \mathbb{R}^{D\times K}$ is the codebook matrix, which contains K codevectors as columns. Then the optimization problem can be formulized as

$$MSE_Q = \min_{\{\boldsymbol{C}\},\{\boldsymbol{B}\}} \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{x}_i - \boldsymbol{C}\boldsymbol{b}_i\|_2^2 \tag{3.9}$$

where $\boldsymbol{B} \in \mathbb{R}^{K\times N}$ is the binary selection matrix. Here the optimization problem reduces to finding the optimal matrices[9] $\boldsymbol{B}$ and $\boldsymbol{C}$, which minimize the quantization error.

[9] The methods using VQ for ANN usually employs more than one quantizer and aim to minimize the quantization error simultaneously in all of them [63].

As mentioned in the beginning of this chapter, VQ-based ANN methods differ from hashing methods by their distance approximations. In hashing approaches, the distance between a query vector $\boldsymbol{q}$ and a database sample $\boldsymbol{p}$ is approximated by the corresponding binary distance between their codes $\boldsymbol{y_p}$ and $\boldsymbol{y_q}$, as given in (3.3). However, in VQ-based ANN, the distance approximation is performed with the help of the codevectors in two ways: *Symmetric* and *Asymmetric* [110]. The symmetric case is more similar to hashing, it requires query $\boldsymbol{q}$ to be quantized to its corresponding reconstructed vector $\widehat{\boldsymbol{q}}$, so the distance between $\boldsymbol{q}$ and $\boldsymbol{p}$ is approximated as

$$\sqrt{\|\boldsymbol{q}-\boldsymbol{p}\|_2^2} \sim \sqrt{\|\widehat{\boldsymbol{q}}-\widehat{\boldsymbol{p}}\|_2^2} \tag{3.10}$$

In the asymmetric case, the query does not have to be quantized and the distance approximation is formulized as

$$\sqrt{\|\boldsymbol{q}-\boldsymbol{p}\|_2^2} \sim \sqrt{\|\boldsymbol{q}-\widehat{\boldsymbol{p}}\|_2^2} \tag{3.11}$$

As observed in (3.10) and (3.11), the symmetric case is expected to be more prone to quantization errors, since quantization is performed both on the database element $\boldsymbol{p}$ and the query $\boldsymbol{q}$ [110]. However, in the asymmetric case, quantization error only applies to database elements. The asymmetric case surpasses the symmetric case in terms of retrieval performance [110], hence it is proposed as the default case in almost all VQ approaches for ANN [63].

In both cases, the reconstruction of database elements is not necessary; instead, a look-up table can be created with precomputed distances of the query vector and all the centroids [110]. For the symmetric case, the look-up table is created only once, consisting of K^2 elements. The elements of the look-up table (LUT) for the symmetric case can be calculated as follows:

$$LUT[k,l] = \sqrt{\|\boldsymbol{C}[k]-\boldsymbol{C}[l]\|_2^2} \tag{3.12}$$

where $LUT[k,l]$ is the distance between the k^{th} codevector $\boldsymbol{C}[k]$ and the l^{th} codevector $\boldsymbol{C}[l]$. In the asymmetric case, the look-up table is created once for each query. The elements of the look-up table for the asymmetric case can be calculated as follows:

$$LUT[k] = \sqrt{\|\boldsymbol{q} - \boldsymbol{C}[k]\|_2^2} \tag{3.13}$$

As the contributions to the ANN literature proposed in this thesis belong to VQ for ANN, the prior art of this branch is investigated in detail in Chapter 3.3.

3.2.4 Comparison of ANN Techniques

So far, ANN techniques have been investigated in three major categories as Partition Tree Structures for ANN, Hashing for ANN and Vector Quantization for ANN. All these methods approach the ANN problem from a different perspective. In this subchapter, the advantages and disadvantages of the aforementioned methods are presented in comparison with each other.

As discussed in Chapter 3.2.1, Partition Tree Structure based methods mainly propose to reorganize the datasets using tree structures. The main concern here is to provide fast access to the (approximate) nearest neighbor. The methods under this category usually keep the dataset as is, i.e., the dataset is not compressed. In addition to the original dataset, a tree structure is created to provide fast access to the samples, which requires additional storage space. The nearest neighbor approximation is improved by using a technique called backtracking (or priority search), where the candidates for the nearest neighbor are collected from the nearest leaves of the tree and re-ranked. If enough leaves are traversed, accessing the actual nearest neighbor can be guaranteed.

Hashing based methods investigated in Chapter 3.2.2 aim to encode dataset samples as binary strings and use binary distance metrics to approximate the distances in the original space. Encoding of the dataset as binary strings provides compression for the dataset, so that a much greater number of samples can be searched at once. Furthermore, binary distance calculations are much faster than calculating distances between vectors in the original space, so a significant increase in speed is also achieved by hashing methods. However, approximation of original distances with binary metrics is inferior and this affects the search performance.

In Chapter 3.2.3, VQ-based approaches for the ANN problem have been discussed. Similar to hashing, VQ-based methods also propose to compress dataset samples using

binary strings. Instead of using binary distance metrics, it is proposed to use look-up tables to approximate the distances. Using look-up tables is still much faster than actual distance calculation, and provides a better approximation compared to binary distances.

In the case of Big Data, two major problems arise from the associated great number samples and dimensions. First is the storage requirement. A dataset is required to be loaded onto RAM before a search can be performed. The larger the dataset, the smaller percentage of it can be loaded to the RAM at once. Loading the dataset to the RAM is a time consuming process, so keeping the number of loads per search to a minimum is recommended. The second problem is the computational complexity of the distance calculation. The greater the vector dimension, the more computations are required in order to calculate the distance between two given samples. The increase in the computational cost of distance calculation is more noticeable when the size of the dataset grows bigger. ANN methods have to be able to handle these problems while providing good approximations.

Comparing the given ANN categories while taking the very large dataset sizes and great number of dimensions into account, Tree based approaches are the least favorable as the hashing and VQ-based approaches provide dataset compressions, which give them a great advantage in terms of storage space requirements. Sometimes the additional space required for the tree structure grows even bigger than the dataset itself [78]. However, hashing and VQ-based approaches provide extreme compressions for the dataset, and the required additional storage space is negligible for very large datasets [63], [64], [110]. In general, VQ-based approaches are more demanding in terms of additional storage space compared to hashing based methods but this amount is still negligible compared to the dataset size [63], [64].

The growth in the vector dimensions also affect the search speed of tree based approaches, as this branch is the most prone to the effects of the “curse of dimensionality” [43], [63], [111]. In some cases of high dimensions, it is stated that tree based approaches can become slower than exact nearest neighbor search [68] [79]. Hamming distance is a very fast distance approximation as it is possible to use binary logical operators, but sometimes it may be required to use look-up tables as well for Hashing methods. This makes their speed comparable to VQ-based methods [110].

In terms of approximation performance, tree-based methods theoretically have the highest upper bound, but this strongly depends on the selection of the backtracking depth. The deeper the backtracking is, the higher the computational requirements. However, since the dataset is kept as is, the perfect reconstruction is still possible. Hashing based approaches suffer from the limited number of possible distances, as a result of using

binary distance metrics. For a $\mathcal{B}$-bit hash code, there are only $\mathcal{B}-1$ different distances. VQ's look-up tables provide much higher distance approximation resolution [110]. This directly affects the retrieval performance. In [110], PQ is compared to SH [86] and FLANN [77], and the PQ shows significant improvement in performance compared to the other two methods. Also in [112], a comparison of OPQ with various hashing algorithms is presented, showing that OPQ outperforms them all with a notable margin.

The comparisons discussed above in terms of storage space, speed and performance are summarized in Table 1 given below. According to these comparisons, VQ-based approaches are the most suitable for Big Data problem. For that reason, in this thesis, the prior art for these methods are investigated extensively in Chapter 3.3, and five new methods in this field have been proposed, which are presented in Chapter 4.

Table 1: Comparison of ANN Techniques

	Storage Requirement	**Computational Complexity**	**Retrieval Performance**	**Exact NN**	**High D**	**Dataset Compression**
Tree	High	High	Low	Yes	No	No
Hashing	Low	Low	Medium	No	Yes	Yes
VQ	Low	Low	High	No	Yes	Yes

3.3 Prior Art in Vector Quantization for ANN

The algorithms using VQ techniques for ANN can be investigated in three major subchapters. The first one consists of the algorithms which are based on a technique called Product Quantization (PQ) [110]. The second subchapter is based on another quantization technique called Residual Vector Quantization (RVQ) [113], while the third includes hybrid methods, which have features borrowed from the first two techniques.

3.3.1 Product Quantization Based Techniques

Product Quantization has been a pioneering method of VQ for ANN, both historically and methodologically. After the first proposal of PQ in 2009 [114], several methods have been inspired from the proposed approach and modifications and variations of PQ have been proposed in the literature. In this subchapter, first the original PQ [110] is discussed in detail. Then its variations such as Transform Coding (TC) [115], Optimized Product Quantization (OPQ) [116] and Cartesian K-Means (CKM) [117] are considered.

3.3.1.1 Product Quantization

The first algorithm that proposes to apply VQ techniques for ANN is Product Quantization (PQ) [110]. PQ is already mentioned in Chapter 2.2, and it is actually one of the first strategies to extend quantization to the multidimensional case [28]. In [110], Jégou *et al*. propose to introduce this approach to the ANN problem. The main motivation behind the idea of using the PQ technique for VQ can be explained with an example. For the popular 128-dimensional image descriptor SIFT [118], a 64-bit quantizer (with a rate of only 0.5 bits per component) contains 2^{64} centroids. This number is too high to be produced by Lloyd's method directly. It requires brute force nearest neighbor search for all the centroids. Even storing that many centroids is not feasible. PQ proposes an efficient solution to this problem by splitting the input vector $\boldsymbol{x} \in \mathbb{R}^D$ into M distinct D/M dimensional subvectors of $\boldsymbol{x}^{(m)} \in \mathbb{R}^{D/M}$ $(1 \leq m \leq M)$. The subspace separation is formulized in (3.14).

$$\begin{aligned} \boldsymbol{x} &= \{\boldsymbol{x}[1], \boldsymbol{x}[2], \dots, \boldsymbol{x}[D]\}^{\boldsymbol{T}} \\ \boldsymbol{x}^{(1)} &= \{\boldsymbol{x}[1], \boldsymbol{x}[2], \dots, \boldsymbol{x}[D/M]\}^{\boldsymbol{T}} \\ \boldsymbol{x}^{(m)} &= \left\{\boldsymbol{x}\left[\frac{(m-1)D}{M} + 1\right], \dots, \boldsymbol{x}[mD/M]\right\}^{\boldsymbol{T}} \end{aligned} \tag{3.14}$$

Since the l_2 distance calculation allows the following calculation:

$$\|\boldsymbol{p} - \boldsymbol{q}\|_2^2 = \sum_{m=1}^{M} \left\|\boldsymbol{p}^{(m)} - \boldsymbol{q}^{(m)}\right\|_2^2 \tag{3.15}$$

Then for each subvector, Lloyd's quantization can be applied separately using M quantizers in total. This provides efficient encoding for vectors with high cardinality. The optimization formula given in (3.9) can be modified for PQ as given in (3.16), where the superscripts in parenthesis refer to orthogonal subspaces.

$$
\begin{aligned}
MSE_Q^{(PQ^{(1)})} &= \min_{\{\boldsymbol{C}^{(1)}\},\{\boldsymbol{B}^{(1)}\}} \frac{1}{N}\sum_{i=1}^{N} \left\| \boldsymbol{x}_i^{(1)} - \boldsymbol{C}^{(1)}\boldsymbol{b}_i^{(1)} \right\|_2^2 \\
MSE_Q^{(PQ^{(2)})} &= \min_{\{\boldsymbol{C}^{(2)}\},\{\boldsymbol{B}^{(2)}\}} \frac{1}{N}\sum_{i=1}^{N} \left\| \boldsymbol{x}_i^{(2)} - \boldsymbol{C}^{(2)}\boldsymbol{b}_i^{(2)} \right\|_2^2 \\
&\vdots \\
MSE_Q^{(PQ^{(M)})} &= \min_{\{\boldsymbol{C}^{(M)}\},\{\boldsymbol{B}^{(M)}\}} \frac{1}{N}\sum_{i=1}^{N} \left\| \boldsymbol{x}_i^{(M)} - \boldsymbol{C}^{(M)}\boldsymbol{b}_i^{(M)} \right\|_2^2
\end{aligned}
\tag{3.16}
$$

$$
MSE_Q^{(PQ)} = \sum_{m=1}^{M} MSE_Q^{(PQ^{(m)})} \tag{3.17}
$$

As observed in (3.16), the quantization problem is divided into M independent subproblems, and each one is solved separately. In other words, for each subspace, a subquantizer is defined, each aiming to achieve the best quantization for their corresponding subspace. Then as shown in (3.17), the total quantization error is found to be the sum of the errors of each subquantizer.

As mentioned above, in PQ, Lloyd's quantization method is selected as subquantizers. For a fixed-rate quantization, the number of subquantizers M and the number of centroids of each Lloyd's quantizer K are related, as expressed below:

$$
\mathcal{B} = M \log_2 K \tag{3.18}
$$

As defined in (3.18), the number of bits $\mathcal{B}$ is defined by the multiplication of the number of subquantizers and the logarithm of the number of centroids per subquantizer. The logarithm term gives the number of bits required to represent K different centroids. According to this relation, if one chooses to have more subquantizers, a smaller number of centroids must be used in each subquantizer. Using (3.18), the number of centroids for a fixed number of bits can be derived as follows:

$$
K = 2^{\mathcal{B}/M} \tag{3.19}
$$

It is suggested in [110] to keep the number of subquantizers to a minimum, as long as the number of centroids is small enough to perform Lloyd's method efficiently. Note that for $M = 1$, the PQ algorithm reduces to Lloyd's method.

As discussed in Chapter 3.2.3, PQ uses Asymmetric Distance to approximate the distance between the query and encoded database elements. The squared distance between a novel sample $\boldsymbol{q}$ and a PQ encoded database element $\widehat{\boldsymbol{p}}$ can be formulized as follows:

$$\|\boldsymbol{q} - \widehat{\boldsymbol{p}}\|_2^2 = \sum_{m=1}^{M} \left\| \boldsymbol{q}^{(m)} - \boldsymbol{C}^{(m)} \boldsymbol{b}_{\widehat{\boldsymbol{p}}}^{(m)} \right\|_2^2 \tag{3.20}$$

PQ uses M separate look-up tables, each of them is obtained using the codebook of a subspace. The look-up tables for PQ are created as given below[10] :

$$LUT^{(m)}[k] = \left\| \boldsymbol{q}^{(m)} - \boldsymbol{C}^{(m)}[k] \right\|_2^2 \tag{3.21}$$

Similar to (3.16) and (3.17), the distance approximation is performed using (3.15) as given below:

$$\|\boldsymbol{q} - \widehat{\boldsymbol{p}}\|_2^2 = \sum_{m=1}^{M} LUT^{(m)} \left[\boldsymbol{b}_{\widehat{\boldsymbol{p}}}^{(m)} \right] \tag{3.22}$$

This means that the distance between a given query and a database point is approximated by M table look-ups and additions. Compared to the actual computational cost of the l_2 distance calculation, which requires D subtractions, multiplications and additions, this is a major improvement [110] $(D \gg M)$. The construction of look-up tables however has a computation complexity of KD. Therefore, the number of database samples must be significantly greater than the number of codevectors per codebook, i.e., $N \gg K$. The number of computations required for encoding of a novel sample is also KD. Even though

[10] Note that for simplicity, the square root term is omitted in look-up tables.

this will be done only once for each database element, if this number is too large, then scaling up the VQ method for very large datasets becomes problematic. Therefore, a VQ algorithm should keep this number as small as possible.

As mentioned in Chapter 2.1, quantization methods are evaluated according to their computational complexities and storage requirements. PQ requires an additional storage of KD floating point numbers to store the codevectors. The look-up table additionally consists of MK numbers. Comparing this to the original size of the dataset, ND, as $N \gg K$, and $D \gg M$, this storage is negligible[11].

3.3.1.2 Transform Coding

Transform Coding (TC), similar to PQ, has its roots reaching to the early days of vector quantization. The idea of applying transform coding technique to vector quantization was proposed first in [31]. Then the same idea is applied to the VQ problem for ANN in 2010 [115]. It states that the assumption of statistically independent vector coefficients usually does not hold. Therefore, when PQ partitions the space into subspaces, because of statistical dependency, inferior codebooks will be generated. TC proposes a solution to this problem by applying a transformation on the vectors before quantization, in order to reduce the statistical dependence. The proposed transformation is Principal Component Analysis (PCA). PCA is an orthogonal transform, which projects given samples onto a new space where the statistical dependence between the coefficients of the vector are minimized [72]. Using the sample covariance matrix, first the eigenvectors are calculated and used to transform the samples[12].

Using PCA transformation before quantization provides statistical independence but the information is distributed unequally among coefficients. These eigenvectors are arranged in a decreasing order according to the eigenvalues, and the eigenvalues already correspond to the sample variances for the new coefficients calculated on the transformed space[13]. In other words, the first dimensions carry more information than the last ones [72]. Hence, if each dimension is encoded using the same number of codes, then the amount of information encoded by the first bits will be more than the last bits, again

---

[11] This number can also be compared to the compressed size of the dataset, $\frac{N\mathcal{B}}{32}$. (A floating-point number is assumed to be stored in 32-bits.)

[12] Note that, eigenvectors are orthogonal to each other, which makes this transformation an orthogonal transformation.

[13] This property of PCA makes it a good approach for dimension reduction. Keeping the higher dimensions and getting rid of the lower dimensions provides minimum information loss.

resulting in inferior codebooks. To solve this problem, TC first proposes to reduce the vector quantization problem into separate scalar quantization problems (similar to PQ, but a special 1-D case). Then it allocates bits proportional to the variance of each principal component, i.e., dimensions corresponding to larger variances are allocated more bits. With this data-driven approach, TC claims that the codebooks fit better to the data, hence improving the quantization error [115]. This approach can be formulized as follows:

$$
\begin{aligned}
MSE_Q^{(TC^{(1)})} &= \min_{\{\boldsymbol{C}^{(1)}\},\{\boldsymbol{B}^{(1)}\}} \frac{1}{N}\sum_{i=1}^{N}\left|\boldsymbol{W}^T\boldsymbol{x}_i[1]-\boldsymbol{C}^{(1)}\boldsymbol{b}_i^{(1)}\right|^2 \\
MSE_Q^{(TC^{(2)})} &= \min_{\{\boldsymbol{C}^{(2)}\},\{\boldsymbol{B}^{(2)}\}} \frac{1}{N}\sum_{i=1}^{N}\left|\boldsymbol{W}^T\boldsymbol{x}_i[2]-\boldsymbol{C}^{(2)}\boldsymbol{b}_i^{(2)}\right|^2 \\
&\vdots \\
MSE_Q^{(TC^{(L)})} &= \min_{\{\boldsymbol{C}^{(L)}\},\{\boldsymbol{B}^{(L)}\}} \frac{1}{N}\sum_{i=1}^{N}\left|\boldsymbol{W}^T\boldsymbol{x}_i[L]-\boldsymbol{C}^{(L)}\boldsymbol{b}_i^{(L)}\right|^2
\end{aligned}
\tag{3.23}
$$

$$
\mathcal{B} = \sum_{l=1}^{L} \log_2 dim\left(\boldsymbol{C}^{(l)}\right) \tag{3.24}
$$

$$
MSE_Q^{(TC)} = \sum_{l=1}^{L} MSE_Q^{(TC^{(l)})} \tag{3.25}
$$

where $dim\left(\boldsymbol{C}^{(l)}\right)$ is the number of centroids on the l^{th} dimension. L is the number of dimensions, to which a bit is assigned. $\boldsymbol{W} \in \mathbb{R}^{D\times L}$ is the transformation matrix obtained from PCA, in which the first L principal components are stored column-wise.

The determination of L, or bit allocation, is another optimization problem. To minimize the total distortion in (3.25), the bits should be allocated to 1-D quantizers so that each assignment reduces the maximum quantization error, keeping in mind that the total number of bits, $\mathcal{B}$, is constant. In [115], it is stated that the optimal solution for the given problem requires computationally prohibitive numerical search. Instead, an approximate solution is proposed. If all components are assumed to be uniformly distributed, then the optimal bit allocation would be achieved if the number of bits on the l^{th} dimension, $\mathcal{B}^{(l)}$, is proportional to the logarithm of the standard deviation, as shown below:

$$dim\left(\boldsymbol{C}^{(l)}\right) = 2^{\mathcal{B}^{(l)}}$$
$$\mathcal{B}^{(l)} \sim \log_2 \sigma^{(l)} \tag{3.26}$$

With these two solutions, TC improves the quantization performance of PQ significantly, especially on datasets where the dimensions exhibit significant statistical dependence [115]. Complexity wise, TC performs better than PQ in both computational and storage requirements. The computational cost of encoding a novel sample for TC is $O(DL + \mathcal{B})$, where $O(DL)$ is the cost of PCA transformation. TC also requires storing the transformation matrix, which requires $O(DL)$ floating numbers. One drawback of TC is that, it heavily relies on the statistical independence claim of PCA transformation. However this claim holds only when the data exhibit Gaussian properties [119].

3.3.1.3 Optimized Product Quantization

Optimized Product Quantization (OPQ) [112], [116] is a variant of PQ. Similar to TC, OPQ also proposes a solution to the statistical dependence problem in PQ. In fact, in [116], two different approaches are proposed: A parametric and a non-parametric approach. In this subchapter, the parametric solution will be investigated. The non-parametric approach is identical with the Cartesian K-Means [117] method if the same initialization is used [112]. Therefore, it will be discussed in the next subchapter, together with Cartesian K-Means.

PQ's efficient solution to vector quantization suffers from the statistical dependence of vector coefficients, as mentioned earlier. An alternative solution to this problem is proposed by OPQ. A transformation is applied to provide statistical independence first. Then similar to PQ, the transformed space is partitioned into subspaces with lower dimensions. The proposed transformation is again PCA. However, as noted for TC, the variances of principal components are not equal. Instead of using different number of bits per component, parametric approach of OPQ proposes to rearrange dimensions so that, the information distributed to each subspace will be balanced. This is provided by an eigenvalue based dimension allocation method. When a dimension is allocated to a subspace, the corresponding eigenvalue, i.e., the variance on that dimension, is added to the subspace's total variance. Starting from the dimension with the highest eigenvalue, each dimension is added to the subspace with the lowest variance. If the maximum number of dimensions for a subspace is reached, then that subspace is skipped. The proposed quantization algorithm requires an orthogonal transformation matrix, which minimizes the statistical dependency between the dimensions, a rotation matrix, which performs the

eigenvalue based allocation, and subquantizers, which are trained separately for each subspace. The parametric OPQ algorithm can be formulized as follows:

$$
\begin{aligned}
MSE_Q^{(OPQ^{(1)})} &= \min_{\{\boldsymbol{C}^{(1)}\},\{\boldsymbol{B}^{(1)}\}} \frac{1}{N}\sum_{i=1}^{N} \left\| (\boldsymbol{R}^T\boldsymbol{W}^T\boldsymbol{x})_i^{(1)} - \boldsymbol{C}^{(1)}\boldsymbol{b}_i^{(1)} \right\|_2^2 \\
MSE_Q^{(OPQ^{(2)})} &= \min_{\{\boldsymbol{C}^{(2)}\},\{\boldsymbol{B}^{(2)}\}} \frac{1}{N}\sum_{i=1}^{N} \left\| (\boldsymbol{R}^T\boldsymbol{W}^T\boldsymbol{x})_i^{(2)} - \boldsymbol{C}^{(2)}\boldsymbol{b}_i^{(2)} \right\|_2^2 \\
&\vdots \\
MSE_Q^{(OPQ^{(M)})} &= \min_{\{\boldsymbol{C}^{(M)}\},\{\boldsymbol{B}^{(M)}\}} \frac{1}{N}\sum_{i=1}^{N} \left\| (\boldsymbol{R}^T\boldsymbol{W}^T\boldsymbol{x})_i^{(M)} - \boldsymbol{C}^{(M)}\boldsymbol{b}_i^{(M)} \right\|_2^2
\end{aligned}
\tag{3.27}
$$

$$
MSE_Q^{(OPQ)} = \sum_{m=1}^{M} MSE_Q^{(OPQ^{(m)})} \tag{3.28}
$$

where $\boldsymbol{W} \in \mathbb{R}^{D\times D}$ is the principal component matrix, and $\boldsymbol{R}$ is the rotation matrix for dimension rearrangement.

As shown in [116], the proposed method improves the quantization performance compared to both TC and PQ. Also in [116], the comparison of OPQ with popular Hashing methods is presented. According to the given results, VQ-based OPQ outperforms hashing methods drastically. Investigating OPQ with respect to complexity, an orthogonal transformation is required for each novel sample to be encoded, which results in an encoding complexity of $O(D^2 + KD)$, where the $O(D^2)$ term belongs to the transformation cost. The same cost was also added as an additional storage requirement, as this transformation matrix needs to be stored.

3.3.1.4 Cartesian K-Means

Cartesian K-Means (CKM) [117] is a very similar method to OPQ [116]. Both have been published in the same conference and the same year. Unlike OPQ, CKM does not inspire from PQ's Cartesian product approach, but rather results in it. CKM formulizes the vector

quantization problem differently than PQ based methods[14]. Each quantization center is formulized as the sum of one codevector from each codebook. This reduces the quantization optimization to the problem given below:

$$MSE_Q^{(CKM)} = \min_{\{\boldsymbol{C}_m\},\{\boldsymbol{B}_m\}} \frac{1}{N}\sum_{i=1}^{N}\left\|\boldsymbol{x}_i - \sum_{m=1}^{M}\boldsymbol{C}_m\boldsymbol{b}_{m_i}\right\|_2^2 \tag{3.29}$$

where $\boldsymbol{C}_m$ is the m^{th} codebook, and $\boldsymbol{b}_{m_i}$ is the binary selection vector for the i^{th} database element which selects from the m^{th} codebook. The solution of the problem given in (3.29) is not trivial. Therefore, further constraints are required. CKM proposes to impose an orthogonality constraint to the subcodebooks, i.e., $\boldsymbol{C}_m^T\boldsymbol{C}_l = \boldsymbol{0}\ if\ m \neq l$. As a result, each codebook matrix is generated from a subspace, which do not intersect with other subspaces. The total codebook matrix $\boldsymbol{C}$ is represented as a multiplication of two matrices: a rotation matrix $\boldsymbol{R}$ with orthonormal columns, and a block diagonal matrix $\boldsymbol{D}$, i.e., $\boldsymbol{C} = \boldsymbol{RD}$

$$\boldsymbol{R} = [\boldsymbol{R}_1\ \boldsymbol{R}_2\ \ldots\ \boldsymbol{R}_M] \quad \boldsymbol{D} = \begin{bmatrix} \boldsymbol{D}_1 & \boldsymbol{0} & \ldots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{D}_2 & & \boldsymbol{0} \\ \vdots & & \ddots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \ldots & \boldsymbol{D}_M \end{bmatrix} \tag{3.30}$$

Therefore, the optimization problem formulated in (3.29) can be rewritten as below, similar to other PQ based methods. Note that, since $\boldsymbol{R}$ is an orthogonal transformation matrix, $\left\|\boldsymbol{R}_m^T\boldsymbol{x}_i - \boldsymbol{D}_m\boldsymbol{b}_{m,i}\right\|_2^2 = \left\|\boldsymbol{x}_i - \boldsymbol{R}_m\boldsymbol{D}_m\boldsymbol{b}_{m,i}\right\|_2^2$ for a full rank $\boldsymbol{R}$, i.e., $\boldsymbol{R}^{\perp} = \emptyset$, where $\boldsymbol{R}^{\perp}$ is the null space of $\boldsymbol{R}$.

[14] Due to its initial formulization, CKM can actually be classified as a Residual Vector Quantization based method. However, by taking the additional constraints applied into account, it is classified as a Product Quantization based algorithm.

$$
\begin{aligned}
MSE_Q^{(CKM_1)} &= \min_{\{\boldsymbol{D}_1\},\{\boldsymbol{R}_1\},\{\boldsymbol{B}_1\}} \frac{1}{N}\sum_{i=1}^{N}\left\|\boldsymbol{R}_1^T\boldsymbol{x}_i - \boldsymbol{D}_1\boldsymbol{b}_{1_i}\right\|_2^2 \\
MSE_Q^{(CKM_2)} &= \min_{\{\boldsymbol{D}_2\},\{\boldsymbol{R}_2\},\{\boldsymbol{B}_2\}} \frac{1}{N}\sum_{i=1}^{N}\left\|\boldsymbol{R}_2^T\boldsymbol{x}_i - \boldsymbol{D}_2\boldsymbol{b}_{2_i}\right\|_2^2 \\
&\vdots \\
MSE_Q^{(CKM_M)} &= \min_{\{\boldsymbol{D}_M\},\{\boldsymbol{R}_M\},\{\boldsymbol{B}_M\}} \frac{1}{N}\sum_{i=1}^{N}\left\|\boldsymbol{R}_M^T\boldsymbol{x}_i - \boldsymbol{D}_M\boldsymbol{b}_{M_i}\right\|_2^2
\end{aligned}
\tag{3.31}
$$

$$
MSE_Q^{(CKM)} = \sum_{m=1}^{M} MSE_Q^{(CKM_m)} \tag{3.32}
$$

CKM proposes an iterative solution for each of these subproblems. Starting with an initial rotation $\boldsymbol{R}_m$, a Lloyd's quantizer is trained, giving a solution for $\boldsymbol{D}_m$ and $\boldsymbol{B}_m$. Then replacing those back into (3.31), the optimal $\boldsymbol{R}_m$ can be obtained by solving the orthogonal Procrustes problem [93]. The iterations continue until a satisfactory convergence is obtained.

Compared to OPQ's parametric model, CKM generates slightly better transformations, as observed in [112]. The difference is less noticeable when the data follows Gaussian properties. CKM shares the same complexity with OPQ, both in computation and additional storage.

3.3.2 Residual Vector Quantization Based Techniques

Residual Vector Quantization (RVQ) is also among the pioneering methods of VQ for ANN. It was proposed right after PQ in 2010 [113]. RVQ inspired several methods in the literature. In this subchapter, first the original RVQ is investigated in detail and then its variants such as Optimized (Enhanced) Residual Vector Quantization (ERVQ) [120], Additive Quantization (AQ) [121] and Composite Quantization (CQ) [122] are examined.

3.3.2.1 Residual Vector Quantization

Similar to PQ, RVQ has its roots back in the early days of vector quantization. The first examples of residual vector quantization was called multi-stage vector quantization, and

was proposed in 1980s [34]. RVQ is very similar to this initial proposal and it adapts the quantization approach to the ANN problem.

Different from VQ for ANN approaches mentioned in Chapter 3.3.1, RVQ trains its codebooks in the original space, i.e., unlike PQ's codebooks, where $\boldsymbol{C}^{PQ} \in \mathbb{R}^{D/M \times K}$, RVQ codebooks $\boldsymbol{C}^{RVQ} \in \mathbb{R}^{M \times K}$. A Lloyd's quantizer is trained for each codebook as in PQ based methods, but each codebook is trained on the residuals of the previous codebook. At its most generic case, the optimization problem can be formulized as follows:

$$MSE_Q^{(RVQ)} = \min_{\{\boldsymbol{C}\},\{\boldsymbol{B}\}} \frac{1}{N} \sum_{i=1}^{N} \left\| \boldsymbol{x}_i - \sum_{m=1}^{M} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \right\|_2^2 \tag{3.33}$$

Note that the equation above is identical to the initial formulization that is used in CKM, which was given in (3.29). However, the optimization problem cannot be simply rewritten as M subproblems, as previously done in CKM, since there is no orthogonality constraint imposed. Solving the problem in (3.33) for all M codebooks at once is also intractable. In order to propose a tractable solution, RVQ brings a hierarchical structure and follows a top-down approach to generate the codebooks. The RVQ optimization problem can be formulated as follows:

$$\begin{aligned}
MSE_Q^{(RVQ_1)} &= \min_{\{\boldsymbol{C}_1\},\{\boldsymbol{B}_1\}} \frac{1}{N} \sum_{i=1}^{N} \left\| \left(\boldsymbol{x}_i - \sum_{m=1}^{1} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \right) \right\|_2^2 \\
MSE_Q^{(RVQ_2)} &= \min_{\{\boldsymbol{C}_2\},\{\boldsymbol{B}_2\}} \frac{1}{N} \sum_{i=1}^{N} \left\| \left(\boldsymbol{x}_i - \sum_{m=1}^{2} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \right) \right\|_2^2 \\
&\vdots \\
MSE_Q^{(RVQ_M)} &= \min_{\{\boldsymbol{C}_M\},\{\boldsymbol{B}_M\}} \frac{1}{N} \sum_{i=1}^{N} \left\| \left(\boldsymbol{x}_i - \sum_{m=1}^{M} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \right) \right\|_2^2
\end{aligned} \tag{3.34}$$

$$MSE_Q^{(RVQ)} = MSE_Q^{(RVQ_M)} \tag{3.35}$$

As it can be seen in (3.34), the optimization problem is divided into M subproblems in a hierarchical manner. First, the top-most layer's codebook is obtained, using a Lloyd's quantizer. Then the corresponding residuals are moved to the next layer for codebook generation. This continues for M layers. Note that, unlike PQ based approaches, the final

quantization error is the quantization error of the last layer, as shown in (3.35). The hierarchical structure of RVQ proposes a simple solution to the codebook generation problem defined in (3.33). The quantization is improved at every layer by quantizing the error of the previous layer.

RVQ provides a better quantization error than PQ, but this comes at an additional cost. Asymmetric distance calculation in RVQ is not as simple as it is in PQ because of the missing orthogonality constraint. The distance between a novel query and an RVQ coded database element can be formulated as follows:

$$\|\boldsymbol{q}-\widehat{\boldsymbol{p}}\|_2^2=\left\|\boldsymbol{q}-\sum_{m=1}^{M}\boldsymbol{C}_m\boldsymbol{b}_{m\widehat{\boldsymbol{p}}}\right\|_2^2 \tag{3.36}$$

Using the distance given in (3.36), it is not possible to calculate it using look-up tables. Without the look-up tables, this calculation is M times more expensive than the distance calculation in the original space. Therefore, (3.36) is modified so that, it is possible to create look-up tables for distance calculation, as given below:

$$\begin{aligned}\|\boldsymbol{q}-\widehat{\boldsymbol{p}}\|_2^2=\left\|\boldsymbol{q}-\sum_{m=1}^{M}\boldsymbol{C}_m\boldsymbol{b}_{m\widehat{\boldsymbol{p}}}\right\|_2^2&=\|\boldsymbol{q}\|_2^2-2\left\langle\boldsymbol{q},\sum_{m=1}^{M}\boldsymbol{C}_m\boldsymbol{b}_{m\widehat{\boldsymbol{p}}}\right\rangle+\left\|\sum_{m=1}^{M}\boldsymbol{C}_m\boldsymbol{b}_{m\widehat{\boldsymbol{p}}}\right\|_2^2\\&=\|\boldsymbol{q}\|_2^2-2\sum_{m=1}^{M}\langle\boldsymbol{q},\boldsymbol{C}_m\boldsymbol{b}_{m\widehat{\boldsymbol{p}}}\rangle+\sum_{m=1}^{M}\sum_{l=1}^{M}\langle\boldsymbol{C}_m\boldsymbol{b}_{m\widehat{\boldsymbol{p}}},\boldsymbol{C}_l\boldsymbol{b}_{l\widehat{\boldsymbol{p}}}\rangle\end{aligned} \tag{3.37}$$

As shown in (3.37), the asymmetric distance is calculated in $\frac{M^2}{2}+M+1$ summations. Note that each dot product term given in (3.37) can be pre-calculated and stored in a look-up table. The first look-up table consists of dot products between the given query and all codevectors. The second look-up table consists of dot products of all codevectors among themselves. Therefore, the first table requires an additional storage of $\mathrm{O}(KM)$ while the second requires $\mathrm{O}(KM^2)$. In addition, RVQ requires the storage of the codevectors, which has a size of $\mathrm{O}(MDK)$. This is also M times larger compared to PQ's $\mathrm{O}(KD)$. As a result, compared to PQ, RVQ requires more calculations and more storage.

3.3.2.2 Optimized Residual Vector Quantization

Optimized Residual Vector Quantization [120] was proposed first with the name Enhanced RVQ (ERVQ) in [123]. As the name suggests ERVQ proposes to optimize RVQ by targeting the independent training of codebooks. A "joint" training scheme is proposed in order to minimize further the quantization error. The algorithm is developed on top of RVQ, and it uses the same approach to perform encoding and distance approximation. Hence, the complexity and storage requirements are the same with RVQ. In other words, ERVQ only aims to propose better codebooks than RVQ.

In ERVQ, it is emphasized that only the last layer of RVQ corresponds to the overall quantization error, as also shown in (3.35). The other layers only consider the quantization error at their own layer, without taking the overall quantization error into account [120]. Therefore, a codebook generation scheme, which considers the overall quantization error at each layer, is proposed by the joint training algorithm. The algorithm initializes the codebooks using RVQ. Instead of holding onto the top-down training approach of RVQ, ERVQ proposes to update upper layer codebooks, keeping the lower layers fixed, in an iterative fashion. Once encoding is performed, the "residuals" corresponding to a given layer are calculated. Then using these residuals, the codebook of that layer is updated, using the same approach in RVQ. The improvement in MSE_Q is observed and if it is below a predefined percentage, the iterations are ended. With this approach, ERVQ obtains a significant increase in performance, compared to RVQ, e.g. using 64-bits with ERVQ produces the same performance with RVQ using 72-bits.

The hierarchical approach proposed by RVQ is a simple solution, but since each layer quantizes the residuals of the previous layer, the improvement in quantization error decreases with every layer. In other words, in terms of quantization error, higher layers have much greater contribution than lower layers, despite the fact that they have the same rate for quantization [120]. This is an important drawback of RVQ, which ERVQ proposes to fix by adding the iterative codebook update stage.

3.3.2.3 Additive Quantization

Proposed in 2014, Additive Quantization (AQ) [121], as the name suggests, propose to quantize vectors as addition of codevectors, similar to RVQ and ERVQ. In other words, the generic formulation of the quantization error minimization problem in AQ is the same as RVQ, as given in (3.33). For the sake of completeness, the formulation for AQ is repeated below:

$$MSE_Q^{(AQ)} = \min_{\{\boldsymbol{C}_m\},\{\boldsymbol{B}_m\}} \frac{1}{N} \sum_{i=1}^{N} \left\| \boldsymbol{x}_i - \sum_{m=1}^{M} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \right\|_2^2 \tag{3.38}$$

Unlike RVQ, AQ does not use a hierarchical structure[15]. This means that, M codevectors will be selected from M different codebooks, with almost no constraints. This relaxed structure allows AQ to obtain much better codebooks for quantization, but also makes the computation extremely demanding [121].

As AQ defines no hierarchy among the codebooks, encoding a novel sample is not as simple as in RVQ. Instead, a heuristic search method is proposed to find the suitable codevectors among all the codebooks. The proposed algorithm is the Beam Search [124]. AQ initializes the beam search by selecting the nearest H codevector candidates among MK codevectors. Then the residuals for each candidate is calculated. For each residual, H more nearest codevectors are selected as candidates among the remaining $(M-1)K$ codevectors, meaning that the codebook of the first codevector candidate is omitted. This results in H^2 candidates in total. In order to prevent the exponential growth of candidates, the top H among the H^2 candidates are picked in terms of the least quantization error. Then again, the residuals are calculated and the nearest codevectors are obtained among the remaining codebooks. This is repeated until M codevectors are selected in total. Since there are MK codevectors in total and H candidates are kept, this operation is computationally very demanding. AQ proposes to use look-up tables in order to decrease the computational requirements.

For codebook learning, AQ also follows the iterative scheme, which updates codes and codebooks sequentially. To update the codebooks, AQ proposes to use a least-square approach. Decomposing the least-square solution for (3.38) into D dimensions, the problem can be written as D least-square problems, which can be formulated as given below:

$$\forall i = 1 \dots N \sum_{m=1}^{M} \boldsymbol{C_m} \boldsymbol{b}_{m_i}[d] = \boldsymbol{x}_i[d] \tag{3.39}$$

[15] Actually, AQ does not even cite RVQ.

where $\boldsymbol{x}_i[d]$ is the d^{th} component of $\boldsymbol{x}_i$. Using (3.39), N equations over MK variables are defined, and each are solved in the least-quadratic sense. After solving these equations, the codebooks are updated. Using the updated codebooks, encoding is repeated for all N samples. The iterations continue until convergence achieved.

As given in (3.37), RVQ requires $\frac{M^2}{2} + M + 1$ additions to compute the distance approximation. In [121], an alternative approach is proposed to reduce this cost to M, which is the cost of PQ based methods. The equation in (3.37) is composed of three parts. The first is the norm of the query vector, the second is the sum of dot products of the query and codevectors, and the third is the sum of the dot products of codevectors among themselves. The third term also corresponds to the norm of the database element. AQ proposes to encode this norm separately, using a scalar quantizer. $M - 1$ layers are used to quantize the given vector and the remaining bits are reserved for the norm quantization. In the experiments given in [121], it is shown that this approach provides slightly worse quantization performance than AQ itself.

AQ outperforms its successors drastically; however, this boost in quantization performance comes with a significant increase in the computational complexity. The computational complexity of encoding with AQ is $\mathrm{O}(M^2K^2(M + \log(MK)) + KD)$. Compared to the complexity of RVQ, which is $\mathrm{O}(MKD)$, this is significantly higher. On the other hand, AQ's additional storage requirement is the same as that of RVQ, which is also $\mathrm{O}(MKD)$.

3.3.2.4 Composite Quantization

Composite Quantization (CQ) [122] is the last of RVQ based methods that are discussed in detail in this chapter. CQ also consists of additive codebooks, but the main motivation behind CQ is to make distance approximation faster. This problem was mentioned in AQ, as the distance approximation given in (3.37) requires $\frac{M^2}{2} + M + 1$ additions. AQ proposed encoding the norm of the database sample using a separate quantizer and reserving extra bits. Contrarily, CQ introduces this as a constraint to its optimization, letting the third term in (3.37) be a constant. When this term is constant, the distance approximation can be calculated in $\mathrm{O}(M)$ operations, without reserving extra bits for quantized norms. The formulation presented by CQ is given as follows:

$$MSE_Q^{(CQ)} = \min_{\{\boldsymbol{C}_m\},\{\boldsymbol{B}_m\},\epsilon} \frac{1}{N}\sum_{i=1}^{N}\left\|\boldsymbol{x}_i - \sum_{m=1}^{M}\boldsymbol{C}_m\boldsymbol{b}_{m_i}\right\|_2^2 + \lambda\sum_{i=1}^{N}\sum_{m=1}^{M}\sum_{l=1,l\neq m}^{M}\left((\boldsymbol{C}_m\boldsymbol{b}_{m_i})^T\boldsymbol{C}_l\boldsymbol{b}_{l_i} - \epsilon\right)^2 \qquad (3.40)$$

where ϵ is a constant, i.e., $\epsilon = \sum_{m=1}^{M} \sum_{l=1,l\neq m}^{M} \left(\boldsymbol{C}_m \boldsymbol{b}_{m_i}\right)^T \boldsymbol{C}_l \boldsymbol{b}_{l_i}$, and λ is the penalty parameter.

Similar to AQ, CQ follows an iterative approach to solve the given optimization problem for $\boldsymbol{C}$, $\boldsymbol{B}$ and ϵ. Fixing $\boldsymbol{B}$ and ϵ, the problem turns into a unconstrained nonlinear optimization problem. CQ proposes to use a quasi-Newton solver to solve this problem. Fixing $\boldsymbol{C}$ and $\boldsymbol{B}$, ϵ can be simply calculated as given below:

$$\epsilon = \frac{1}{N} \sum_{i=1}^{N} \sum_{m=1}^{M} \sum_{l=1,l\neq m}^{M} \left(\boldsymbol{C}_m \boldsymbol{b}_{m_i}\right)^T \boldsymbol{C}_l \boldsymbol{b}_{l_i} \tag{3.41}$$

Fixing $\boldsymbol{C}$ and ϵ, CQ encodes the samples using the Iterative Conditional Modes (ICM) algorithm. ICM is very similar to codebook improvement proposed in ERVQ. Fixing all the codes but one, the left out code is selected as the nearest among the codevectors in the corresponding codebook. Then another code is updated the same way, keeping the rest fixed. The iterations continue until the maximum number of iterations reached.

The complexity of ICM encoding depends on the number of iterations allowed. CQ keeps this number very low, only three iterations, hence encoding of a novel sample requires $O(3MKD)$ computations. And additional storage requirement is the same as RVQ and ERVQ, $O(MKD)$. CQ successfully accelerates the distance approximation, with a slight decrease in performance compared to AQ.

3.3.3 Hybrid Techniques

As discussed so far, two major branches of approaches are used in VQ for ANN. The first one, PQ based approaches; aims to partition the original space into subspaces and train separate quantizers in each subspace. The final quantization is obtained as the Cartesian products of the subspaces. The second branch, RVQ based approaches; performs the quantization of a sample by adding multiple codevectors, which belong to the same space with the given sample. The basic methods and their variants have been discussed in detail in Chapter 3.3.1 and Chapter 3.3.2. In this chapter, techniques, which benefit from PQ based and RVQ based approaches will be discussed. Locally Optimized Product Quantization (LOPQ) [125], Additive Product Quantization (APQ) [121], Optimized Cartesian K-Means (OCKM) [126] and (Optimized) Tree Quantization (OTQ) [127] are the methods, which will be discussed in this chapter in detail.

3.3.3.1 Locally Optimized Product Quantization

Locally Optimized Product Quantization (LOPQ) [125], as its name suggests, is a variant of OPQ [112], which was discussed in detail in Chapter 3.3.1.3 . LOPQ proposes to train locally optimized OPQs in order to improve the quantization error. The main motivation behind generating local OPQs stems from inverted indexing methods such as Inverted File with Asymmetric Distance Computation (IVFADC)[110] and Inverted Multi-Index (IMI) [128]. These methods propose to implement quantization using a coarse quantizer first, in order to provide inverted file indexing. It is possible to retrieve a subset of a dataset by using the inverted indices. Then ANN can be performed only on this subset. This is called non-exhaustive search [110]. Several quantization algorithms also propose non-exhaustive versions of their methods. For example, RVQ and PQ propose using a Lloyd's quantizer as a coarse quantizer and their own quantization method to quantize the residuals of the coarse quantizer. In IMI, PQ is used as both the coarse quantizer and the subquantizer. Differently from the mentioned approaches, LOPQ uses a Lloyd's quantizer as the coarse quantizer; however, it proposes to train a different subquantizer for each code obtained from the coarse quantizer.

In [110], IVFDAC first trains the coarse quantizer, calculates the residuals for each vector and then trains one single PQ subquantizer using all the residuals. On the contrary, LOPQ groups the samples according to their nearest codevector in the coarse quantization stage, and then trains an OPQ per group. This is equivalent to cluster the data using K-means and quantizing each cluster with a unique OPQ [125].

LOPQ improves the quantization performance of OPQ but this improvement comes with a noteworthy price in terms of storage requirements. Compared to OPQ's storage requirement of $O(D^2 + KD)$, LOPQ requires $\varkappa$ times more space to store the codevectors, $O\big(\varkappa(KD + D^2)\big)$, where $\varkappa$ is the number of codevectors of the coarse quantizer[16]. It also requires $\varkappa$ look-up tables, one for each subquantizer. LOPQ is more expensive than OPQ in terms of computational costs. LOPQ requires $O(\varkappa D + D^2 + KD)$ operations, $\varkappa D$ more than OPQ.

3.3.3.2 Additive Product Quantization

The Additive Product Quantization (APQ) is a hybrid variant of AQ, which was explained in detail in Chapter 3.3.2.3. As discussed above the Beam Search algorithm used in the

[16] Note that $\varkappa \geq K$, so this is a very important increase.

encoding step of AQ is computationally expensive, and grows cubically with the number of layers M. An alternative hybrid approach to AQ is proposed in [121] in order to decrease this computational cost.

Decomposing M as $M = M_1 M_2$, APQ first splits the dimensions of original space into M_1 subspaces (As previously done in OPQ) and quantizes each subspace using with AQ using M_2 codebooks. In other words, APQ is a Cartesian product of M_2 AQs, trained in subspaces of $\mathbb{R}^{D/M_2}$. This provides a better computational complexity for the encoding step with a slight drop in quantization performance. Compared to the computational complexity of $\mathrm{O}(M^2K^2(M + \log(MK)) + KD)$ of AQ, APQ has a complexity of $\mathrm{O}\left(D^2 + \frac{M}{M_2}(M_2^2K^2(M_2 + \log(M_2K)) + KD)\right)$. Using OPQ to split the original space into subspaces requires also a transformation matrix, and this adds an additional storage requirement of D^2 to AQ's requirement of $\mathrm{O}(MKD)$, which results in $\mathrm{O}(D^2 + MKD)$ The authors in [121] suggest using APQ instead of AQ when $M > 4$.

3.3.3.3 Optimized Cartesian K-Means

Optimized Cartesian K-Means (OCKM) [126] is a hybrid variant of the Cartesian K-Means [117], which was discussed in detail in Chapter 3.3.1.4. Similar to PQ based approaches, OCKM also partitions the original space into subspaces and the final quantization is a Cartesian product of subquantizers. Furthermore, similar to CKM, OCKM rotates the data before subspace partitioning. However, OCKM proposes two codebooks for each subspace and picking two codevectors and adding them up. The results of these additions are concatenated and the quantized vector is obtained. The optimization problem formulation in OCKM can be presented as follows:

$$MSE_Q^{(OCKM)} = \min_{\{\boldsymbol{C}_m\},\{\boldsymbol{B}_m\},\boldsymbol{R}} \frac{1}{N}\sum_{i=1}^{N}\left\|\boldsymbol{x}_i - \boldsymbol{R}\begin{bmatrix}\sum_{c=1}^{2}\boldsymbol{C}_c^{(1)}\boldsymbol{b}_{c_i}^{(1)}\\ \vdots \\ \sum_{c=1}^{2}\boldsymbol{C}_c^{(M/2)}\boldsymbol{b}_{c_i}^{(\mathrm{M}/2)}\end{bmatrix}\right\|_2^2 \tag{3.42}$$

where $\boldsymbol{C}_c^{(m)}$ is the c^{th} codebook on the m^{th} subspace. The codebooks that OCKM generates in different subspaces are orthogonal to each other. However, for the codebooks that are generated within the same subspace, there is no constraint defined. As mentioned in AQ, the solution of this problem requires complex computations. In order to

reduce this complexity, OCKM favors orthogonal codebooks to codebooks in the same subspace, i.e., they keep the number of codebooks per subspace limited to 2.

OCKM follows an iterative approach that is similar to the one in CKM to solve the optimization problem given in (3.42). First, the rotation matrix $\boldsymbol{R}$ is obtained fixing the codebooks and binary codes. Then the codebooks are updated fixing the binary codes and the rotation matrix by the least squares, as also proposed in AQ. Finally, fixing the codebooks and rotation, first the encodings are determined. The method proposed for encoding is also Beam Search, similar to AQ. But since the number of codebooks is only two, the encoding is much faster [126].

The motivation behind OCKM is similar to APQ, but while OCKM selects two codebooks per subspace, APQ divides the space into two subspaces. Therefore, the codebooks trained in OCKM have stronger orthogonality constraints compared to APQ. This can be observed in the quantization performance as well. While OCKM outperforms CKM, it cannot reach to the performance of APQ. In terms of computational complexity, however, OCKM is the winner. The number of computations required for encoding in OCKM is approximately $O(HKD)$, where H is the number of selected codevectors for beam search. OCKM is also better with respect to the additional storage space requirements, which is $O(D^2 + 2KD)$.

3.3.3.4 (Optimized) Tree Quantization

(Optimized) Tree Quantization (OTQ) [127] is the last hybrid method that is discussed in this chapter. It is an improvement over AQ, mainly targeting the high computational cost of encoding. As mentioned in Chapter 3.3.2.3, AQ imposes no constraints to its codebooks, hence finding the optimum encoding is computationally demanding. Here in OTQ, a structure is imposed on the codebooks, based on a tree graph. The vertices of the tree correspond to the codebooks and each dimension is assigned to an edge. Hence, each codebook is trained on a space, which is spanned by only the dimensions that are assigned to the incident edges of its vertex. The optimization problem can be formulized as follows:

$$MSE_Q^{(OTQ)} = \min_{\{\boldsymbol{C}\},\{\boldsymbol{B}\},\{e\}} \sum_{i=1}^{N} \sum_{d=1}^{D} \sum_{m=1}^{M} \sum_{l=m}^{M} e(m,l) \left| \boldsymbol{C}_m \boldsymbol{b}_{m_i}[d] + \boldsymbol{C}_l \boldsymbol{b}_{l_i}[d] - \boldsymbol{x}_i[d] \right|^2 \tag{3.43}$$

Here $e(m,l)$ describes the graph connections, i.e., $e(m,l) = 1$ if dimension d is assigned to edge (m,l).

Instead of using beam search, with the help of the given structure, OTQ proposes to use dynamic programming (max-product algorithm) to optimize the encoding process. The codebook learning of OTQ is iterative, similar to the other proposed algorithms. Firstly, keeping the codes $\boldsymbol{B}$ and graph connections e fixed, the optimization problem given in (3.43) turns into a sum of errors for independent dimensions, which can be reorganized as a least squares problem. Solving this problem for $\boldsymbol{C}$ gives the updated codebooks [127]. Secondly, keeping $\boldsymbol{C}$ and $\boldsymbol{B}$ fixed, the problem reduces to a minimization over assignment variables, which is solved using an Integer Linear Programming (ILP) solver. Finally keeping $\boldsymbol{C}$ and e fixed, the problem is the encoding problem, which is solved by the max product algorithm for all training samples, as mentioned in the beginning of this chapter. Furthermore, OTQ proposes to include OPQ's optimal rotation in the beginning so that each sample is rotated before quantization[17].

OTQ outperforms AQ in quantization performance. In addition, with the help of the imposed structure, OTQ changes the beam search of AQ with max product algorithm, which brings a boost to the encoding efficiency. OTQ's encoding requires $\mathrm{O}(D^2 + KD + MK^2)$ computations, which is significantly less than AQ's requirement of $\mathrm{O}(M^2K^2(M + \log(MK)) + KD)$. OTQ also requires slightly more additional storage space than AQ to store the codevectors. The storage requirement of OTQ is $\mathrm{O}(D^2 + MKD)$.

3.3.4 Comparison of Prior Art in Vector Quantization for ANN

So far, in Chapters 3.3.1, 3.3.2 and 3.3.3 several VQ-based ANN methods have been investigated in detail. In this chapter, a comparison among those methods is provided. The comparison of VQ-based methods can be performed in three aspects: quantization performance, computational complexity for encoding and distance approximation and additional storage requirements.

To compare the aforementioned methods, two popular benchmark datasets are used, SIFT1M and GIST1M [110], which are constructed in three parts: learning, base and query. The learning set is designed for training of the quantizers. The base set is the largest of all sets, which will be encoded and stored. The query set is the set of samples, of which the nearest neighbors from the base set are retrieved. SIFT1M consists of SIFT descriptors [118]. The learning set of SIFT1M includes 100.000 samples obtained from

[17] This is the reason for the "Optimized" term in parenthesis in the beginning of the name of the quantization method.

Flickr, and the base set, which consists of 1 Million samples; and the query set, which consists of 10.000 samples, are from INRIA Holiday images dataset [129]. Similarly, GIST1M consists of GIST descriptors [130]. The learning set of GIST1M includes 500.000 samples obtained from Tiny Image dataset [131], and the base, which consists of 1 Million samples; and query set, which consists of 1000 samples, are again from Flickr and INRIA Holiday images dataset [129].

In all the methods discussed above, the retrieval performances are presented using these two datasets. During the tests, different sizes of encodings are investigated, from 16-bits to 128-bits. 32-bits and 64-bits are the most common ones; hence, the comparison among the methods is presented using these two lengths. To measure the performance of the quantizer, *recall@R* is proposed [110] and used in all of the methods given above. This metric measures the percentage of query vectors, of which the Euclidean nearest neighbor is retrieved correctly in the first *R* rankings[18]. Usually the metric is calculated for R=1, 10 and 100 [110]. The performances of the methods mentioned above are given below, as presented in the original publications:

Table 2: Performance Comparison for Prior-Art Methods in VQ for ANN

TEST RESULTS FOR SIFT1M, 32-BIT CODES

	recall@1	recall@10	recall@100
PQ	0.052	0.230	0.595
TC	0.057	0.197	0.519
RVQ	*NA*	*NA*	*NA*
CKM/OPQ	0.068	0.273	0.658
AQ	0.106	**0.415**	**0.825**
CQ	*NA*	*NA*	*NA*
E-LOPQ	**0.134**	0.385	0.738
OCKM	*NA*	0.348	0.742
ERVQ	*NA*	*NA*	*NA*
OTQ	0.093	0.368	0.793

TEST RESULTS FOR GIST1M, 32-BIT CODES

	recall@1	recall@10	recall@100
PQ	0.023	0.068	0.176
TC	0.053	0.104	0.291
RVQ	*NA*	*NA*	*NA*
CKM/OPQ	0.054	0.142	0.396
AQ	**0.069**	**0.189**	**0.467**
CQ	*NA*	*NA*	*NA*
E-LOPQ	0.049	0.131	0.362
OCKM	*NA*	0.172	**0.467**
ERVQ	*NA*	*NA*	*NA*
OTQ	*NA*	*NA*	*NA*

TEST RESULTS FOR SIFT1M, 64-BIT CODES

	recall@1	recall@10	recall@100
PQ	0.224	0.599	0.924
TC	0.205	0.535	0.877
RVQ	0.257	0.659	0.952
CKM/OPQ	0.243	0.638	0.940
APQ	0.298	0.741	**0.972**
CQ	0.288	0.716	0.967
E-LOPQ	0.297	0.703	0.957
OCKM	0.274	0.680	0.945
ERVQ	0.276	0.694	0.962
OTQ	**0.317**	**0.748**	**0.972**

TEST RESULTS FOR GIST1M, 64-BIT CODES

	recall@1	recall@10	recall@100
PQ	0.076	0.218	0.504
TC	0.096	0.223	0.547
RVQ	0.113	0.325	0.676
CKM/OPQ	0.118	0.334	0.715
AQ/APQ	*NA*	*NA*	*NA*
CQ	**0.135**	**0.377**	**0.729**
E-LOPQ	0.116	0.331	0.656
OCKM	0.130	0.358	0.720
ERVQ	0.115	0.341	0.711
OTQ	*NA*	*NA*	*NA*

[18] When R=1, the metric corresponds to “precision”.

As presented above, Table 2 compares the nearest neighbor retrieval performance of the prior-art methods in VQ for ANN[19]. Surprisingly; the best *recall@1* is obtained by a different method for each dataset. For *recall@10* and *recall@100* AQ gives the best performance for 32-bit encoding. OTQ outperforms the other methods for 64-bit encoding for SIFT1M, while CQ is the best method for the same length on GIST1M.

The computational complexity of encoding for the aforementioned methods are presented in Table 3.

Table 3: Comparison of Computational Costs of Encoding

Method	**Cost of Encoding**	**Cost of Encoding for Different Datasets and Code Lengths (Number of Operations)**			
		SIFT1M-32	**SIFT1M-64**	**GIST1M-32**	**GIST1M-64**
PQ	$O(KD)$	32768	32768	245760	245760
TC	$O(DL+\mathcal{B})$	**2592**	**5184**	**19232**	**38464**
RVQ	$O(MKD)$	131072	262144	983040	1966080
CKM/OPQ	$O(D^2+KD)$	49152	49152	1167360	1167360
AQ	$O(M^2K^2(M+log(MK))+KD)$	14712832	79724544	14925824	79937536
APQ	$O\left(D^2+\frac{M}{4}(4^2K^2(4+log(4K))+KD)\right)$	14729216	14761984	15847424	16093184
CQ	$O(3MKD)$	393216	786432	2949120	5898240
E-LOPQ	$O(KD+KD+D^2)$	81920	81920	1413120	1413120
OCKM	$O(10KD)$	327680	327680	2457600	2457600
ERVQ	$O(MKD)$	131072	262144	983040	1966080
OTQ	$O(D^2+KD+MK^2)$	311296	573440	1429504	1691648

$\boldsymbol{K}$: number of sub-codewords	256	256	256	256
$\boldsymbol{D}$: number of dimensions	128	128	960	960
$\boldsymbol{M}$: number of sub-codebooks	4	8	4	8
$\boldsymbol{L}$: number of reduced dimensions (on average)	20	40	20	40
$\boldsymbol{\mathcal{B}}$: number of bits used for encoding.	32	64	32	64

As shown in Table 3, TC is the fastest method in terms of encoding and AQ is by far the slowest. However, the numbers presented here must be considered together with the retrieval performances given in Table 2, in order to have a proper evaluation for the methods. Finally, Table 4 compares the additional storage requirements of the prior-art methods. As it can be seen, TC requires the least amount of additional storage space, while LOPQ is by far the most demanding one.

[19] E-LOPQ is the exhaustive implementation of LOPQ. The results presented in [125] are different from the ones presented here as the authors did not include the localization overhead in the number of bits.

Table 4: Comparison of Additional Storage Requirements

Method	Cost of Encoding	Storage Cost for Different Datasets and Code Lengths (MB)			
		SIFT1M-32	SIFT1M-64	GIST1M-32	GIST1M-64
PQ	$O(KD)$	0.25	0.25	1.88	1.88
TC	$O(DL)$	**0.02**	**0.04**	**0.15**	**0.29**
RVQ	$O(MKD)$	1.00	2.00	7.5	15
CKM/OPQ	$O(D^2 + KD)$	0.38	0.38	8.91	8.91
AQ	$O(MKD)$	1.00	2.00	7.5	15
APQ	$O(D^2 + MKD)$	1.13	2.13	14.53	22.03
CQ	$O(MKD)$	1.00	2.00	7.5	15
E-LOPQ	$O\big(K(KD + D^2)\big)$	96.00	96.00	2280.00	2280.00
OCKM	$O(D^2 + 2KD)$	0.63	0.63	10.78	10.78
ERVQ	$O(MKD)$	1.00	2.00	7.5	15
OTQ	$O(D^2 + MKD)$	1.13	2.13	14.53	22.03
$\boldsymbol{K}$: number of sub-codewords		256	256	256	256
$\boldsymbol{D}$: number of dimensions		128	128	960	960
$\boldsymbol{M}$: number of sub-codebooks		4	8	4	8
$\boldsymbol{L}$: number of reduced dimensions (on average)		20	40	20	40

To summarize, in Chapter 3.3, the prior techniques in VQ for the ANN problem have been investigated in detail, discussing the main motivation behind every algorithm, including the novelties they propose in order to improve on the previous work. In 3.3.4, the final subchapter of Chapter 3.3, these methods are compared using quantitative measures. In the next chapter, Chapter 4, the contributions to the literature proposed in this thesis will be explained in detail.

4 Contributions

This thesis proposes five new methods for ANN, following a vector quantization approach. The methods in this thesis aim to outperform their predecessors or improve the encoding complexity and/or the additional storage requirements. The contributions are first divided into two subchapters as Hybrid Techniques and RVQ based Techniques, similar to the classification given in Chapter 3.3. First, the Hybrid methods will be explained as they were proposed before RVQ based ones.

As required by the compendium type of doctoral thesis followed here, this chapter aims to only summarize the main methods, their merits and their performances with respect to the state-of-the-art, while the details are included in the enclosed publications in the Appendix and referred in the thesis as [P1], [P2], etc.

4.1 Hybrid Techniques

In this thesis, two Hybrid techniques are proposed. The first one is called M-PCA Binary Embedding (MPCA-E) [P1] and the second one is K-Subspaces Quantization (KSSQ) [P2]. In the following subchapters, both of these methods are discussed and their contributions are summarized.

4.1.1 M-PCA Binary Embedding for ANN

Several methods in VQ literature benefit from PCA and the resulting transformation for variance equalization or statistical dependence minimization. The first examples are TC, OPQ and APQ, while CKM and OCKM also use PCA for initializations. However, the performance of PCA in removing statistical dependence in real life data may not be as high as desired since PCA relies on second-order statistics only, i.e., $E\left[\boldsymbol{x}[i]\boldsymbol{x}[j]\right] = 0$. A real-life dataset, which does not satisfy the Gaussian distribution properties, may have significant third or higher order statistical dependencies present, i.e., $E\left[\boldsymbol{x}[i]\boldsymbol{x}[j]\boldsymbol{x}[k]\right] \neq 0$.

A well-known method to improve the performance of PCA is Local PCA [119]. This is an iterative method, which aims to cluster the dataset into $\mathcal{K}$ affine subspaces, with the aim that each resulting cluster will be more Gaussian-like than the whole dataset itself. The samples are assigned to their nearest subspaces, and for each cluster, a new affine subspace is calculated using PCA. The distance of a sample $\boldsymbol{x}$ to an affine subspace $\boldsymbol{\mathcal{F}}$, which is defined by matrix $\boldsymbol{R}$ and the affine shift vector $\boldsymbol{\mu}$, can be formulated as follows:

$$d(\boldsymbol{x}, \boldsymbol{\mathcal{F}}) = \left\| \boldsymbol{R}^{\perp T}(\boldsymbol{x} - \boldsymbol{\mu}) \right\|_2^2 \tag{4.1}$$

where $\boldsymbol{R}^{\perp}$ is the null space of $\boldsymbol{R}$. Furthermore, the Local PCA can be formulated as an optimization problem as follows:

$$\min_{\{\boldsymbol{R}_k\},\{\boldsymbol{\mu}_k\},\{\boldsymbol{\delta}_k\}} \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{\mathcal{K}} \boldsymbol{\delta}_{k,i} \left\| \boldsymbol{R}_k{}^{\perp T}(\boldsymbol{x}_i - \boldsymbol{\mu}_k) \right\|_2^2 \tag{4.2}$$

where

$$\begin{array}{ll} \boldsymbol{\delta}_{k,i} = 1 & \boldsymbol{iff} \quad d(\boldsymbol{x}_i, \boldsymbol{\mathcal{F}}_k) \leq d(\boldsymbol{x}_i, \boldsymbol{\mathcal{F}}_\ell) \; \boldsymbol{for} \; \ell \in \{1,2,\dots,\mathcal{K}\} \\ \boldsymbol{\delta}_{k,i} = 0 & otherwise \end{array} \tag{4.3}$$

The Local-PCA tries to solve the optimization problem given in (4.2) for $\boldsymbol{R}_k$, $\boldsymbol{\mu}_k$ and $\boldsymbol{\delta}_k$ in an iterative way. Keeping $\boldsymbol{R}_k$, $\boldsymbol{\mu}_k$ fixed, it solves for $\boldsymbol{\delta}_k$ by simply assigning each sample to its closest subspace. Then keeping $\boldsymbol{\delta}_k$ fixed, $\boldsymbol{R}_k$ becomes the PCA transformation matrix and $\boldsymbol{\mu}_k$ is the mean value.

In M-PCA Binary Embedding [P1], several embedding methods, which use PCA as a dimension reduction technique such as TC, Iterative Quantization (ITQ) [93] and PCA-Embedding (PCA-E) [99] are investigated. It is proposed to replace the PCA with Local-PCA, and show that this enhances the performance. For a given sample, the quantizer on each subspace is used to generate an embedding candidate and the one with the minimum error is chosen. Therefore, the first bits of encoding are reserved for the index of the quantizer that is selected. Among the aforementioned methods, MPCA-E using TC provides the best results, see [P1] for additional details.

As shown in [P1], the quantization performance is consistently improved when Local PCA (or Multiple PCA) is used instead of a single PCA. The best performance is obtained when TC is used. The performance of MPCA-E is tested in the datasets described in Chapter 3.3.4, and the results are provided in Table 5 and Table 6 [P1]. As expected, having multiple transformations instead of one brings extra computations and requires more additional storage space. For comparison, the results of MPCA-E are presented together with TC. TC is a special case of MPCA-E, where the number of affine subspaces $\mathcal{K} = 1$.

Table 5: M-PCA Embedding Test Results

TEST RESULTS FOR SIFT1M, 32-BIT CODES

	recall@1	recall@10	recall@100
TC	0.057	0.197	0.519
MPCA-E	**0.124**	**0.404**	**0.784**

TEST RESULTS FOR GIST1M, 32-BIT CODES

	recall@1	recall@10	recall@100
TC	0.053	0.104	0.291
MPCA-E	**0.054**	**0.149**	**0.345**

TEST RESULTS FOR SIFT1M, 64-BIT CODES

	recall@1	recall@10	recall@100
TC	0.205	0.535	0.877
MPCA-E	**0.286**	**0.710**	**0.923**

TEST RESULTS FOR GIST1M, 64-BIT CODES

	recall@1	recall@10	recall@100
TC	0.096	0.223	0.547
MPCA-E	**0.110**	**0.312**	**0.662**

Table 6: Computational and Storage Costs of MPCA-E

Method	**Encoding Cost**	**Encoding Cost for Different Datasets and Code Lengths (Number of Operations)**			
		SIFT1M-32	**SIFT1M-64**	**GIST1M-32**	**GIST1M-64**
TC	$O(DL + \mathcal{B})$	**2592**	**5184**	**19232**	**38464**
MPCA-E	$O(2\mathcal{K}DL + \mathcal{K}\boldsymbol{\mathcal{B}})$	1318912	2637824	1229824	2459648
Method	**Storage Cost**	**Storage Cost for Different Datasets and Code Lengths (MB)**			
		SIFT1M-32	**SIFT1M-64**	**GIST1M-32**	**GIST1M-64**
TC	$O(DL)$	**0.02**	**0.04**	**0.15**	**0.29**
MPCA-E	$O(\mathcal{K}DL)$	5.00	10.00	4.69	9.38
$\boldsymbol{D}$: number of dimensions		128	128	960	960
$\boldsymbol{L}$: number of reduced dimensions		20	40	20	40
$\boldsymbol{\mathcal{K}}$: number of subspaces		256	256	32	32
$\boldsymbol{\mathcal{B}}$: number of bits for encoding.		32	64	32	64

Table 6 shows the storage and computational costs for MPCA-E. A novel sample is transformed onto each subspace and this requires $O(\mathcal{K}DL)$ operations. The distance of the novel sample to the subspace also plays an important role on the error so this needs to be calculated as well. (4.1) can be modified as given below so that it will be computationally more efficient when $rank(\boldsymbol{R}^{\perp}) \gg rank(\boldsymbol{R})$:

$$d(\boldsymbol{x}, \boldsymbol{\mathcal{F}}) = \|(\boldsymbol{x} - \boldsymbol{\mu}) - \boldsymbol{R}\boldsymbol{R}^{T}(\boldsymbol{x} - \boldsymbol{\mu})\|_2^2 \qquad (4.4)$$

The distance calculation in (4.4) brings an additional computation of $O(\mathcal{K}DL)$. Therefore, the final computational cost of encoding is $O(2\mathcal{K}DL + \mathcal{K}\mathcal{B})$. As all the transformation matrices for all the subspaces are required to be stored, the additional storage space is $O(\mathcal{K}DL)$.

As it can be seen in Table 5 and Table 6, the proposed method performs significantly better than TC, but it also requires much more computational complexity and additional storage space. More results of MPCA-E can be found in [P1] and [P2]. The comparison of MPCA-E and the prior art are discussed in detail in Chapter 4.3, where the experimental results and cost analysis are presented comparatively in Table 16, Table 17 and Table 18.

4.1.2 K-Subspaces Quantization for ANN

Similar to [P1], K-Subspaces Quantization for ANN [P2] is another method proposed in this thesis, which aims to partition the original space into $\mathcal{K}$ affine subspaces and perform quantization afterwards. As discussed in [P1], already many methods in the literature use a transformation or projection onto a new subspace, where there is either dimension reduction, rotation or reordering. In [P2], a quantizer scheme is proposed, where the dimension reduction and the quantization are optimized jointly.

The quantization error for a quantizer, which uses an orthogonal dimension reduction transformation, can be formulized as follows:

$$MSE_Q = \frac{1}{N}\sum_{i}^{N}\left(\|\boldsymbol{R}^T\boldsymbol{x}_i - Q(\boldsymbol{R}^T\boldsymbol{x}_i)\|_2^2 + \left\|\boldsymbol{R}^{\perp T}\boldsymbol{x}_i\right\|_2^2\right) \tag{4.5}$$

where Q is the quantizer, $\boldsymbol{R}$ is an orthogonal transformation matrix with $\boldsymbol{R} \in \mathbb{R}^{D\times L}$ and $L \leq D$. $\boldsymbol{R}^{\perp}$ spans the orthogonal complement of the range space of $\boldsymbol{R}$. This error can be investigated in two terms. The first term comes from the quantizer and the quantization itself, while the second term is a dimension reduction error, which is independent from the quality of quantization. As mentioned in Chapter 4.1.1, in order to reduce the second term of the error, there are alternative approaches to using a single PCA transformation for dimension reduction in the literature [119], [132], [133]. They propose partitioning the space into subspaces, using more than one PCA for each subspace generated.

Let $\boldsymbol{\mathcal{F}}_{\mathcal{k}} \in \mathbb{R}^{L_{\mathcal{k}}}$ represent a subspace which is defined by the affine shift vector $\boldsymbol{\mu}_{\mathcal{k}} \in \mathbb{R}^{D}$ and the projection matrix $\boldsymbol{R}_{\mathcal{k}} \in \mathbb{R}^{D\times L_{\mathcal{k}}}$. Let $\boldsymbol{X}_{\mathcal{k}} \subset \boldsymbol{X}$ be a cluster of samples such that $\cup_{\mathcal{k}=1}^{\mathcal{K}} \boldsymbol{X}_{\mathcal{k}} = \boldsymbol{X}$ and $\boldsymbol{X}_{\mathcal{k}} \cap \boldsymbol{X}_{\mathcal{j}} = \emptyset$ where $\mathcal{k} \neq \mathcal{j}$. Using the given variables, we obtain the following mean square error by modifying (4.5) for multiple affine subspaces:

$$MSE_Q = \frac{1}{N}\sum_{\hbar=1}^{\mathcal{K}}\sum_{\boldsymbol{x}_i \in \boldsymbol{X}_\hbar}\left(\left\|\boldsymbol{R}_\hbar^T(\boldsymbol{x}_i - \boldsymbol{\mu}_\hbar) - Q_\hbar\left(\boldsymbol{R}_\hbar^T(\boldsymbol{x}_i - \boldsymbol{\mu}_\hbar)\right)\right\|_2^2 + \left\|\boldsymbol{R}_\hbar^{\perp T}(\boldsymbol{x}_i - \boldsymbol{\mu}_\hbar)\right\|_2^2\right) \quad (4.6)$$

Here, note that (4.6) is an extension of (4.2) to the quantization problem. The minimization of the quantization error given in (4.6) is an NP-Hard problem so an approximate iterative solution is proposed. First $\boldsymbol{X}_\hbar$ is initialized using K-Means. Keeping $\boldsymbol{X}_\hbar$ fixed, $\boldsymbol{\mu}_\hbar$ is calculated as the mean of $\boldsymbol{X}_\hbar$, i.e., $\boldsymbol{\mu}_\hbar = E[\boldsymbol{X}_\hbar]$. Then $\boldsymbol{R}_\hbar$ is the PCA transformation matrix calculated for $\boldsymbol{X}_\hbar$. The subquantizer $Q_\hbar$ is trained using the transformed samples $\boldsymbol{R}_\hbar^T(\boldsymbol{X}_\hbar - \boldsymbol{\mu}_\hbar)$. At the end of the first round of the iterations, the clusters are updated for each $\boldsymbol{x}_i$ using the following equation:

$$\boldsymbol{x}_i \in \boldsymbol{X}_{\dot{\hbar}} \; where \;\; \dot{\hbar} = \underset{\hbar}{\operatorname{argmin}}\left(\left\|\boldsymbol{R}_\hbar^T(\boldsymbol{x}_i - \boldsymbol{\mu}_\hbar) - Q_\hbar\left(\boldsymbol{R}_\hbar^T(\boldsymbol{x}_i - \boldsymbol{\mu}_\hbar)\right)\right\|_2^2 + \left\|\boldsymbol{R}_\hbar^{\perp T}(\boldsymbol{x}_i - \boldsymbol{\mu}_\hbar)\right\|_2^2\right) \quad (4.7)$$

In this way, each sample $\boldsymbol{x}_i$ is assigned to the cluster, where the minimum quantization error is obtained. After the update of the clusters, iterations continue until a convergence is achieved or a maximum number of iterations is reached.

As discussed in [P1], using multiple subspaces instead of one brings additional computational and storage requirements. Since there are $\mathcal{K}$ subspaces, it is required to compute $\mathcal{K}$ transformations and store $\mathcal{K}$ matrices. Also since there are $\mathcal{K}$ subquantizers, each of them needs additional storage space. The calculation of the quantization error in each quantizer also brings significant computational cost. Such factors have to be taken into account while selecting a suitable subquantizer. First of all, a subquantizer which benefits from dimension reduction would decrease the computational cost of transformation from $O(\mathcal{K}D^2)$ to $O(\mathcal{K}DL)$. Same applies to the additional storage space, where a full rank transformation would require $O(\mathcal{K}D^2)$ numbers to be stored, a low rank transformation would require $O(\mathcal{K}DL)$.

With these constraints, it is proposed to use a tailored variant of TC as it is the best in terms of computational costs and storage requirements. The bit allocation method in TC is slightly modified for KSSQ, allowing it to reserve more bits to higher dimensions. This causes an increase in dimension reduction, but since the error acquired from dimension reduction is minimized by the proposed joint iterative training scheme, this leads to an improved quantization error in total.

To encode a novel sample in KSSQ, first the quantization error for each subquantizer should be calculated. Then the encoding performed by the subquantizer, which provides the minimum error, is returned as the final encoding. In order to accelerate this process, KSSQ aims to keep the number of candidate subquantizers to a minimum. To select the best candidates, first the distance between the sample and the affine shift vectors $\|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|$ is calculated. The subquantizers which correspond to the nearest affine shift vectors are selected as candidates. This is controlled by an additional parameter $\kappa < \mathcal{K}$. So the encoding cost is reduced from $O(2\mathcal{K}DL + \mathcal{K}\boldsymbol{\mathcal{B}})$ to $O(\mathcal{K}D + 2\kappa DL + \kappa\boldsymbol{\mathcal{B}})$. [20]

As discussed in detail in [P2], KSSQ outperforms the prior-art while providing comparable computational cost and additional storage space requirements. The proposed subspace approach is shown to decrease the quantization error, while increasing the nearest neighbor retrieval performance. The test results of KSSQ on the datasets described in Chapter 3.3.4 are provided in Table 7, while the computational and storage costs are presented in Table 8. The experimental results and the cost analysis of KSSQ are presented in comparison with the prior art in Table 16, Table 17 and Table 18, in Chapter 4.3.

Table 7: KSSQ Test Results

TEST RESULTS FOR SIFT1M, 32-BIT CODES

	recall@1	recall@10	recall@100
MPCA-E	0.124	0.404	0.784
KSSQ	**0.145**	**0.434**	**0.802**

TEST RESULTS FOR GIST1M, 32-BIT CODES

	recall@1	recall@10	recall@100
MPCA-E	0.054	0.149	0.345
KSSQ	**0.078**	**0.191**	**0.437**

TEST RESULTS FOR SIFT1M, 64-BIT CODES

	recall@1	recall@10	recall@100
MPCA-E	0.286	0.710	0.923
KSSQ	**0.325**	**0.754**	**0.976**

TEST RESULTS FOR GIST1M, 64-BIT CODES

	recall@1	recall@10	recall@100
MPCA-E	0.110	0.312	0.662
KSSQ	**0.136**	**0.396**	**0.741**

[20] Note that $O(2\mathcal{K}DL + \mathcal{K}\boldsymbol{\mathcal{B}})$ is the cost of MPCA-E.

Table 8: Computational and Storage Costs of KSSQ

Method	Encoding Cost	Encoding Cost for Different Datasets and Code Lengths (Number of Operations)			
		SIFT1M-32	**SIFT1M-64**	**GIST1M-32**	**GIST1M-64**
MPCA-E	$O(2\mathcal{K}DL + \mathcal{K}\boldsymbol{\mathcal{B}})$	1318912	2637824	1229824	2459648
KSSQ	$O(\mathcal{K}D + 2\kappa DL + \kappa\mathcal{B})$	**115200**	**197632**	**338176**	**645632**
Method	**Storage Cost**	**Storage Cost for Different Datasets and Code Lengths (MB)**			
		SIFT1M-32	**SIFT1M-64**	**GIST1M-32**	**GIST1M-64**
MPCA-E	$O(\mathcal{K}DL)$	5.00	10.00	4.69	9.38
KSSQ	$O(\mathcal{K}DL)$	5.00	10.00	4.69	9.38
$\boldsymbol{D}$: number of dimensions		128	128	960	960
$\boldsymbol{L}$: number of reduced dimensions[21]		20	40	20	40
$\boldsymbol{\mathcal{K}}$: number of subspaces		256	256	32	32
$\boldsymbol{\kappa}$: number of candidate quantizers		16	16	8	8
$\boldsymbol{\mathcal{B}}$: number of bits for encoding		32	64	32	64

4.2 Residual Vector Quantization based Techniques

In this thesis, three RVQ based techniques are proposed. The first one is called Self-Organized Binary Encoding (SOBE) [P3], the second one is Joint K-Means Quantization (JKM) [P4], and the last one is Competitive Quantization (CompQ) [P5]. In the following subchapters, all of these methods are discussed and their performance is summarized.

4.2.1 Self-Organized Binary Encoding for ANN

The hierarchical scheme proposed in RVQ is an efficient solution for codebook generation and codevector selection. As discussed in Chapter 3.3.2.1, thanks to its hierarchy, RVQ divides the codebook generation problem into subproblems, each of which can be solved efficiently and easily using Lloyd's quantizer. Yet this approach has some disadvantages. The hierarchical structure forces each layer to be solved separately and this may lead to convergence to a local minimum for each layer. The inferiority of each layer affects its lower layers, as the following layers quantize the residuals of the previous layers. This makes overfitting a very important problem in RVQ.

[21] The numbers presented here are the average number of dimensions for the subspaces of KSSQ, which are on par with the predefined number of dimensions for MPCA-E

As discussed in Chapter 2.2, clustering techniques have been adapted to VQ problem quite many times. A popular clustering method is Self-Organizing Maps (SOM) [134]. SOM is a neural network, which is used to map the high-dimensional distribution of samples onto a predefined low dimensional grid. It is inspired from the structure of the brain cells, as the neighboring brain cells are discovered to respond to inputs together. SOM usually forms a two dimensional grid of neurons and automatically reorganizes the samples and their association with these neurons. Training of SOM can be interpreted as a competitive learning algorithm, as the weights of the neurons are updated iteratively according to an error measure [134]. SOM brings the concept of "winner neuron" to the neural nets, and updates the weights of the winner neurons and some of its neighbors' weights. With this feature, it can be said that SOM's are more robust against overfitting [135].

In order to improve the performance of RVQ by eliminating the overfitted centroids, in [P3], it is proposed to adapt SOM as a VQ for each layer of RVQ. The definition of SOM starts with the definition of the winner neuron. For an input $\boldsymbol{x}$, the winner neuron is defined as given below:

$$\dot{k} = \underset{k}{\operatorname{argmin}}(\|\boldsymbol{x} - \boldsymbol{w}_k\|_2^2) \tag{4.8}$$

In this way, a winner neuron's weight vector provides the minimum squared error to the given input. SOM proposes to train the weights by using a stochastic gradient descent approach, to minimize the error between a given sample and the corresponding winner neuron. The iterative weight update equation for the winner neuron is given below:

$$\boldsymbol{w}_{\dot{k}}(t+1) = \boldsymbol{w}_{\dot{k}}(t) - \gamma(t)\nabla_{\boldsymbol{w}_{\dot{k}}}(\|\boldsymbol{x} - \boldsymbol{w}_{\dot{k}}\|^2) \tag{4.9}$$

where $\nabla_{\boldsymbol{w}_{\dot{k}}}$ is the gradient operation and $\gamma(t)$ is the learning rate. The gradient can be calculated as:

$$\nabla_{\boldsymbol{w}_{\dot{k}}}(\|\boldsymbol{x} - \boldsymbol{w}_{\dot{k}}\|_2^2) = 2(\boldsymbol{w}_{\dot{k}} - \boldsymbol{x}) \tag{4.10}$$

SOM also updates the neighbors of the winner neuron. Let $\mathcal{N}_{\boldsymbol{w}_{\dot{k}}}$ represent the set of neighbors of the winner neuron. The weight update is performed as shown below:

$$\boldsymbol{w}_k(t+1) = \begin{cases} \boldsymbol{w}_k(t) - \gamma(t)(\boldsymbol{w}_k - \boldsymbol{x}) & if\ k \in \mathcal{N}_{\boldsymbol{w}_k} \\ \boldsymbol{w}_k(t) & else \end{cases} \tag{4.11}$$

Usually neighbors are defined by a 2-D Gaussian kernel, but alternatively the distance between the neurons can also be used to define the neighborhood and control the neighboring weight updates [134]. In [P3], a similar approach is followed. A number of nearest neurons are defined as the neighboring neurons. However, since SOM usually defines a 2-dimensional grid, a multidimensional SOM grid is proposed in [P3]. A transform coding based clustering is applied to initialize the neurons positions and weights. Then for each neuron, a number of nearest neurons are assigned as neighbors. The neighboring neurons are updated in relation to their distances to the winner neuron. The neighborhood relation is preserved throughout the iterations.

In [P3], an improvement for the encoding algorithm is also proposed. RVQ's encoding is simple and layer based as it simply selects the nearest codevectors to residuals. SOBE proposes to optimize the encoding iteratively, by keeping $M-1$ codevectors fixed and updating the m^{th} one for all M codevectors. This continues until the decrease in the quantization error converges or a maximum number of iterations is reached.

Together with SOM based codebook generation and improved encoding scheme, SOBE outperforms RVQ on tests performed on benchmark datasets of ANN. The test results of SOBE are provided in Table 9 and the computational and storage costs are presented in Table 10. SOBE is presented in comparison with the prior art in Table 16, Table 17 and Table 18, in Chapter 4.3.

Table 9: SOBE Test Results

TEST RESULTS FOR SIFT1M, 32-BIT CODES

	recall@1	recall@10	recall@100
RVQ	NA	NA	NA
SOBE	**0.100**	**0.348**	**0.731**

TEST RESULTS FOR GIST1M, 32-BIT CODES

	recall@1	recall@10	recall@100
RVQ	NA	NA	NA
SOBE	**0.064**	**0.189**	**0.403**

TEST RESULTS FOR SIFT1M, 64-BIT CODES

	recall@1	recall@10	recall@100
RVQ	0.257	0.653	0.946
SOBE	**0.282**	**0.701**	**0.962**

TEST RESULTS FOR GIST1M, 64-BIT CODES

	recall@1	recall@10	recall@100
RVQ	0.113	0.325	0.676
SOBE	**0.136**	**0.360**	**0.705**

Table 10: Computational and Storage Costs of SOBE

Method	Encoding Cost	Encoding Cost for Different Datasets and Code Lengths (Number of Operations)			
		SIFT1M-32	**SIFT1M-64**	**GIST1M-32**	**GIST1M-64**
RVQ	$O(MKD)$	**131072**	**262144**	**983040**	**1966080**
SOBE	$O(2MKD)$	262144	524288	1966080	3932160
Method	**Storage Cost**	**Storage Cost for Different Datasets and Code Lengths (MB)**			
		SIFT1M-32	**SIFT1M-64**	**GIST1M-32**	**GIST1M-64**
RVQ	$O(MKD)$	1.00	2.00	7.5	15
SOBE	$O(MKD)$	1.00	2.00	7.5	15
$\boldsymbol{M}$: number of layers		128	128	960	960
$\boldsymbol{K}$: number of codevectors		256	256	256	256
$\boldsymbol{D}$: number of dimensions		8	8	4	4

As shown in Table 10, the iterative encoding converges in about two iterations on average, so the cost of encoding for SOBE is twice the cost of RVQ. Both methods require the same amount of additional storage space. More details about SOBE can be found in [P3].

4.2.2 Joint K-Means Quantization for ANN

Another RVQ based VQ method for ANN proposed in this thesis is Joint K-Means Quantization (JKM). As mentioned in Chapter 3.3.2.1 and 4.2.1 RVQ's hierarchical structure separates the quantization problem into M subproblems and the solution of each of problem strongly depend on the previous one. However, in its proposed solution, RVQ does not consider this dependence. ERVQ claims to offer a joint training scheme, but the proposed algorithm only provides an update on the codebooks generated by RVQ, which are already obtained independently from each other. Hence, the proposed codebook update does not really construct a joint scheme.

Nevertheless, a combination of the hierarchical structure with a joint codebook generation strategy would increase the performance while enjoying the low encoding complexity. Following this claim, Joint K-Means is proposed [P4]. JKM expands the "K-means"[22] training on one of RVQ's layers to all layers, providing a joint training scheme. Investigating the training scheme of K-Means[23], first an "expectation" step is performed, where

[22] K-Means clustering algorithm and Lloyd's vector quantization are sometimes used interchangeably in the literature.
[23] K-Means can be interpreted as an *"Expectation-Maximization"* approach.

the vectors are assigned to the nearest codevectors. Later a “maximization” step follows the expectation step where the codevectors are updated with the means of the assigned vectors. RVQ applies these steps for many iterations separately at each layer. In JKM, it is proposed to extend this to all layers.

The “expectation-maximization” steps of JKM occurs as follows: in the “expectation” step, each vector is assigned to its *“selected”* codevector and the residual is immediately calculated and transferred to the next layer, where the same operation will be repeated until the final layer is reached. Then in the maximization stage, codevectors at each layer are updated with the means of assigned codevectors. Therefore, while RVQ waits for the quantization on a layer to converge, JKM propagates the residuals through layers during the iterations. Note that, JKM does not assign the given vector to the nearest codevector, but instead it assigns to the *“selected”* codevector, and this selection is performed by the encoding algorithm. JKM proposes a joint encoding algorithm, which takes also the layer below the current layer into account, while selecting the codevector from the current layer. Incorporating this encoding method into the training improves the codebook generation even further.

Encoding in RVQ is also performed independently for each layer in a nearest neighbor fashion. In other words, the nearest codevector from the corresponding codebook is selected for each residual. However, this does not guarantee the minimum error. Let $\boldsymbol{c}_{1,a}$ be the closest codevector to $\boldsymbol{x}$ and $\boldsymbol{c}_{2,a}$ is the closest codevector to the first residual $\boldsymbol{r}_1 = \boldsymbol{x}\text{-}\boldsymbol{c}_{1,a}$. $\boldsymbol{c}_{1,b}$ is a different codevector from the first codebook, i.e., $\boldsymbol{c}_{1,a} \neq \boldsymbol{c}_{1,b}$ and $\boldsymbol{c}_{2,b}$ is a codevector from the second codebook. The suboptimality of this encoding scheme can be proven as follows:

lemma: Given $\|\boldsymbol{x} - \boldsymbol{c}_{1,a}\|_2^2 \leq \|\boldsymbol{x} - \boldsymbol{c}_{1,b}\|_2^2$, and $\|(\boldsymbol{x} - \boldsymbol{c}_{1,a}) - \boldsymbol{c}_{2,a}\|_2^2 \leq \|(\boldsymbol{x} - \boldsymbol{c}_{1,a}) - \boldsymbol{c}_{2,b}\|_2^2$ there exist at least one $\boldsymbol{c}_{1,b}$ and $\boldsymbol{c}_{2,b}$, which satisfy

$$\|(\boldsymbol{x} - \boldsymbol{c}_{1,a}) - \boldsymbol{c}_{2,a}\|_2^2 \geq \|(\boldsymbol{x} - \boldsymbol{c}_{1,b}) - \boldsymbol{c}_{2,b}\|_2^2 \tag{4.12}$$

proof: Assume that $\boldsymbol{x} = \boldsymbol{c}_{1,b} + \boldsymbol{c}_{2,b}$. Then (4.12) turns into the following:

$$\|(\boldsymbol{x} - \boldsymbol{c}_{1,a}) - \boldsymbol{c}_{2,a}\|_2^2 \geq 0 \tag{4.13}$$

which is always true. Now if one can show that the assumption for $\boldsymbol{x} = \boldsymbol{c}_{1,b} + \boldsymbol{c}_{2,b}$ is valid, the proof is complete. If $\boldsymbol{x} = \boldsymbol{c}_{1,b} + \boldsymbol{c}_{2,b}$, then putting it in the first inequality given in lemma gives the following:

$$\left\|\boldsymbol{c}_{1,b} + \boldsymbol{c}_{2,b} - \boldsymbol{c}_{1,a}\right\|_2^2 \leq \left\|\boldsymbol{c}_{2,b}\right\|_2^2 \tag{4.14}$$

Rearranging the terms in (4.14), one can obtain the equation below:

$$\left\|\boldsymbol{c}_{2,b} - \left(\boldsymbol{c}_{1,a} - \boldsymbol{c}_{1,b}\right)\right\|_2^2 \leq \left\|\boldsymbol{c}_{2,b}\right\|_2^2 \tag{4.15}$$

which is true when $\left\|\boldsymbol{c}_{1,a} - \boldsymbol{c}_{1,b}\right\|_2^2 \leq 2\langle \boldsymbol{c}_{2,b}, \boldsymbol{c}_{1,a} - \boldsymbol{c}_{1,b}\rangle$. For the second inequality in lemma, when the proposed assumption for $\boldsymbol{x} = \boldsymbol{c}_{1,b} + \boldsymbol{c}_{2,b}$ is put into the inequality, then the following inequality is obtained:

$$\left\|\left(\boldsymbol{c}_{1,b} + \boldsymbol{c}_{2,b}\right) - \boldsymbol{c}_{1,a} - \boldsymbol{c}_{2,a}\right\|_2^2 \leq \left\|\boldsymbol{c}_{1,b} - \boldsymbol{c}_{1,a}\right\|_2^2 \tag{4.16}$$

Rearranging the terms in (4.16), one can obtain the equation below:

$$\left\|\left(\boldsymbol{c}_{1,b} - \boldsymbol{c}_{1,a}\right) - \left(\boldsymbol{c}_{2,a} - \boldsymbol{c}_{2,b}\right)\right\|_2^2 \leq \left\|\boldsymbol{c}_{1,b} - \boldsymbol{c}_{1,a}\right\|_2^2 \tag{4.17}$$

which is true when $\left\|\left(\boldsymbol{c}_{2,a} - \boldsymbol{c}_{2,b}\right)\right\|_2^2 \leq 2\langle \boldsymbol{c}_{1,b} - \boldsymbol{c}_{1,a}, \boldsymbol{c}_{2,a} - \boldsymbol{c}_{2,b}\rangle$. Since (4.15) and (4.17) can be true according to the selection of codevectors, in other words they are not always false, then $\boldsymbol{x} = \boldsymbol{c}_{1,b} + \boldsymbol{c}_{2,b}$ is a valid case, hence the proof is complete.

In order to improve the encoding performance, *"joint encoding"* is proposed in JKM. Joint encoding is similar to beam search in AQ or OCKM, but much less complex since it enjoys the hierarchical structure, which reduces the number of required computations significantly. The joint encoding method searches for the codevector with the minimum quantization error in a small neighborhood of the nearest codevector. So instead of the nearest codevector, it is proposed to select the H nearest codevectors and calculate the residuals for each of them. Then the same operation is repeated for each residual, giving H^2 candidates. The best H according to the quantization error is selected and the oper-

ations proceed until the final layer is reached. To explain the computational costs of encoding in detail, the distance between the m^{th} layer residual $\boldsymbol{r}_m$ of the given vector $\boldsymbol{x}$, and the k^{th} codevector on the m^{th} layer $\boldsymbol{c}_{m,k}$ can be rewritten as follows:

$$\begin{aligned} d(\boldsymbol{r}_m, \boldsymbol{c}_{m,k}) &= \left\| \boldsymbol{x} - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l - \boldsymbol{c}_{m,k} \right\|_2^2 \\ &= \left\| \boldsymbol{x} - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l \right\|_2^2 - 2\left\langle \boldsymbol{x} - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l , \boldsymbol{c}_{m,k} \right\rangle + \left\| \boldsymbol{c}_{m,k} \right\|_2^2 \\ &= \left\| \boldsymbol{x} - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l \right\|_2^2 - 2\langle \boldsymbol{x}, \boldsymbol{c}_{m,k} \rangle + 2 \sum_{l=1}^{m-1} \langle \dot{\boldsymbol{c}}_l, \boldsymbol{c}_{m,k} \rangle + \left\| \boldsymbol{c}_{m,k} \right\|_2^2 \end{aligned} \tag{4.18}$$

where $\dot{\boldsymbol{c}}_l$ is the nearest codevector on the l^{th} layer. For each layer, note that the first term is already calculated in the previous layers. The third and fourth terms can be retrieved from a look-up table. Hence, the second term should be calculated first for all the codevectors, which requires $O(KD)$ operations for one layer. The look-ups for the third and fourth terms require $O(mKH)$ look-ups and additions for the m^{th} layer. Finally, among all the distances the best H are selected, which cost $O(KH \log H)$. This is repeated M times so the final cost of encoding is $O\left(MDK + \frac{(M-1)(M-2)}{2} KH + MKH \log H\right)$. More details on this encoding scheme can be found in [P4] and [P5].

To conclude, JKM takes the lower layers into account during both codebook generation and vector encoding steps. This affects the quantization performance as expected. The tests on ANN benchmarks are shown in Table 11 and Table 12. JKM is also presented in comparison with the prior art in Table 16, Table 17 and Table 18, in Chapter 4.3.

Table 11: JKM Test Results

TEST RESULTS FOR SIFT1M, 32-BIT CODES

	recall@1	recall@10	recall@100
SOBE	0.100	0.348	0.731
JKM	**0.121**	**0.402**	**0.790**

TEST RESULTS FOR GIST1M, 32-BIT CODES

	recall@1	recall@10	recall@100
SOBE	0.064	0.189	0.403
JKM	**0.077**	**0.213**	**0.511**

TEST RESULTS FOR SIFT1M, 64-BIT CODES

	recall@1	recall@10	recall@100
SOBE	0.282	0.701	0.962
JKM	**0.323**	**0.759**	**0.980**

TEST RESULTS FOR GIST1M, 64-BIT CODES

	recall@1	recall@10	recall@100
SOBE	0.136	0.360	0.705
JKM	**0.140**	**0.413**	**0.769**

Table 12: Computational and Storage Costs of JKM

Method	Encoding Cost	Encoding Cost for Different Datasets and Code Lengths (Number of Operations)			
		SIFT1M-32	**SIFT1M-64**	**GIST1M-32**	**GIST1M-64**
SOBE	$O(2MKD)$	**262144**	**524288**	1966080	3932160
JKM	$O\left(MDK + \frac{(M-1)(M-2)}{2}KH + MKH\log H\right)$	319488	761856	**1171456**	**2465792**

Method	Storage Cost	Storage Cost for Different Datasets and Code Lengths (MB)			
		SIFT1M-32	**SIFT1M-64**	**GIST1M-32**	**GIST1M-64**
SOBE	$O(MKD)$	1.00	2.00	7.5	15
JKM	$O(MKD)$	1.00	2.00	7.5	15

$\boldsymbol{M}$: number of layers	128	128	960	960
$\boldsymbol{K}$: number of codevectors	256	256	256	256
$\boldsymbol{D}$: number of dimensions	8	8	4	4
$\boldsymbol{H}$: number of candidates	32	32	32	32

4.2.3 Competitive Quantization for ANN

The last method proposed in this thesis is Competitive Quantization (CompQ) [P5]. As discussed in Chapter 3.3.2.1, 4.2.1 and 4.2.2 RVQ suffers from suboptimality in its codebook generation approach. In Chapter 4.2.2, JKM [P4] has shown that a joint codebook training scheme may improve the quantization performance. In this proposed method, it is aimed to take the joint codebook generation in JKM one step further. In [P4], JKM extended the iterations of K-Means to multiple quantization layers. This approach allowed higher level codebooks to take lower level codebooks into account. However, the codevectors are still updated sequentially in JKM, from top to bottom. As shown in (3.33), the quantization error is produced by the codevectors of all layers together, so all of them should be updated simultaneously, to minimize this error. In [P4], a joint optimization for codebook generation is proposed using the stochastic gradient descent. The codevectors are updated using the gradient of the error as shown below:

$$\dot{\boldsymbol{c}}_m(t+1) = \dot{\boldsymbol{c}}_m(t) - \gamma_m(t)\nabla_{\dot{\boldsymbol{c}}_m}\left(\left\|\boldsymbol{x} - \sum_{l=1}^{M}\dot{\boldsymbol{c}}_l\right\|_2^2\right) \tag{4.19}$$

where $\nabla_{\dot{\boldsymbol{c}}_m}$ is the gradient operation and $\gamma_m(t)$ is the learning rate of the m^{th} layer, which decreases with time. $\dot{\boldsymbol{c}}_m$ is the selected codevector for the sample $\boldsymbol{x}$ for the m^{th} layer. The error gradient can be calculated as follows:

$$\nabla_{\dot{c}_m}\left(\left\|\boldsymbol{x}-\sum_{l=1}^{M}\dot{\boldsymbol{c}}_l\right\|_2^2\right)=2\left(\sum_{l=1}^{M}\dot{\boldsymbol{c}}_l-\boldsymbol{x}\right) \tag{4.20}$$

Therefore, each selected codevector for a sample $\boldsymbol{x}$ can be updated as given in the equation below, moving the corresponding codevector closer towards the input sample.

$$\dot{\boldsymbol{c}}_m(t+1)=\dot{\boldsymbol{c}}_m(t)+2\gamma_m(t)\left(\boldsymbol{x}-\sum_{l=1}^{M}\dot{\boldsymbol{c}}_l\right) \tag{4.21}$$

CompQ proposes to have a smaller learning rate for lower layers, since the upper layers correspond to larger quantization error. As discussed in Chapter 4.2.2, the nearest codevector encoding at each level can be suboptimal, and in order to improve that a joint encoding method is proposed in [P4]. In [P5] CompQ also uses the same encoding scheme, and incorporates it into its training. In other words, CompQ updates the codevector selected by the encoding algorithm. With this codebook generation approach and encoding scheme, CompQ obtains the best quantization performance. The tests on ANN benchmarks presented in Chapter 3.3.4 are shown in Table 13. The computational cost and additional storage requirements of CompQ are identical with JKM, which are presented in detail in Chapter 4.2.2 Table 12. CompQ is also presented in comparison with the prior art methods in Table 16, Table 17 and Table 18, in Chapter 4.3. More detailed discussion on CompQ can be found in [P5].

Table 13: CompQ Test Results

TEST RESULTS FOR SIFT1M, 32-BIT CODES

	recall@1	recall@10	recall@100
JKM	0.121	0.402	0.790
CompQ	**0.135**	**0.435**	**0.818**

TEST RESULTS FOR GIST1M, 32-BIT CODES

	recall@1	recall@10	recall@100
JKM	**0.077**	**0.213**	**0.511**
CompQ	0.072	0.200	0.504

TEST RESULTS FOR SIFT1M, 64-BIT CODES

	recall@1	recall@10	recall@100
JKM	0.323	0.759	0.980
CompQ	**0.352**	**0.795**	**0.987**

TEST RESULTS FOR GIST1M, 64-BIT CODES

	recall@1	recall@10	recall@100
JKM	0.140	0.413	0.769
CompQ	**0.155**	**0.419**	**0.801**

In [P5], CompQ also emphasizes an inherent property of the hierarchical structure of quantization, which can be used for non-exhaustive search. Non-exhaustive versions of PQ and OPQ using an additional coarse quantization layer are already present in the literature [110]–[112], [125], [128]. However, CompQ can perform non-exhaustive search without using an additional coarse quantizer, thanks to the hierarchical structure. The

upper layers of CompQ can be used as a coarse quantizer in order to improve the total search speed. In [P5], it is shown that the same quantization performance can be obtained with almost 9 times less comparisons, using the non-exhaustive search property of CompQ.

Table 14: CompQ Non-Exhaustive Search Test Results

NON-EXHAUSTIVE SEARCH, SIFT1M 64-BIT CODES

	Avg. No. Comparisons	Avg. Speed-Up	recall		
			@1	@10	@100
Non-Exhaustive	108064	x 9	0.352	0.795	0.986
Exhaustive	1000000	x 1	0.352	0.795	0.988

The performance of non-exhaustive search with CompQ is also tested against other non-exhaustive search methods. The test is performed on a very large scale dataset, SIFT1B, which consists of 1 Billion samples [110]. CompQ outperforms the other methods such as LOPQ [125], IVFADC [110] and non-exhaustive adaptation of OPQ (IOPQ) [112], [125], with a slight increase in the total database storage[24].

Table 15: CompQ Non-Exhaustive Search Performance Comparison

NON-EXHAUSTIVE SEARCH SIFT1B 80-BIT CODES

	#bits	recall@1	recall@10	recall@100
IVFADC	77	0.088	0.372	0.733
I-OPQ	77	0.114	0.399	0.777
LOPQ	77	0.199	0.586	0.909
CompQ	80	**0.222**	**0.626**	**0.914**

4.3 Comparison of the Contributions with Prior Work

The proposed methods in this thesis for the problem of VQ-based ANN have been discussed in Chapter 4.1 and 4.2. In this chapter the proposed methods are compared with the prior art. The first table, Table 16, compares the quantization performance of the

[24] The compared results are presented in [125], where 77-bit codes are selected. To match that CompQ is trained with $M = 10$ layers, each layer consisting of 256 codevectors (8-bit). 3-bit increase in representation of each database elements increases the total size of SIFT1B by only 3.9%.

proposed methods to the prior work. The second table, Table 17, presents the computational costs and finally the third table, Table 18, compares the additional storage requirements.

Being an intermediate method, MPCA-E [P1] shows that introducing more than one subspace improves the performance of quantization. MPCA-E's improvement over TC was shown in Table 5, and in Table 16, it can be observed that MPCA-E outperforms other methods such as PQ, OPQ/CKM, OCKM, RVQ and ERVQ. KSSQ [P2] is shown to borrow the main idea of MPCA-E and apply further optimizations to improve the performance as it produces the best results for *recall@1*[25] at 32-bit coding. SOBE [P3] is the first proposed method targeting the suboptimality of RVQ and it is shown to perform better than RVQ itself in Table 9.

[25] recall@1 is considered as the most important metric among the three metrics used here [110], where it measures the success rate of retrieving the real nearest neighbor on top of the ranked list.

Table 16: Comparison of Proposed Methods with Prior Work in Performance

TEST RESULTS FOR SIFT1M, 32-BIT CODES

	recall@1	recall@10	recall@100
PQ	0.052	0.230	0.595
TC	0.057	0.197	0.519
RVQ	*NA*	*NA*	*NA*
CKM/OPQ	0.068	0.273	0.658
AQ	0.106	0.415	**0.825**
CQ	*NA*	*NA*	*NA*
E-LOPQ	0.134	0.385	0.738
OCKM	*NA*	0.348	0.742
ERVQ	*NA*	*NA*	*NA*
OTQ	0.093	0.368	0.793
MPCA-E	0.124	0.404	0.784
KSSQ	**0.145**	0.434	0.802
SOBE	0.100	0.348	0.731
JKM	0.121	0.402	0.790
CompQ	0.135	**0.435**	0.818

TEST RESULTS FOR GIST1M, 32-BIT CODES

	recall@1	recall@10	recall@100
PQ	0.023	0.068	0.176
TC	0.053	0.104	0.291
RVQ	*NA*	*NA*	*NA*
CKM/OPQ	0.054	0.142	0.396
AQ	0.069	0.189	0.467
CQ	*NA*	*NA*	*NA*
E-LOPQ	0.049	0.131	0.362
OCKM	*NA*	0.172	0.467
ERVQ	*NA*	*NA*	*NA*
OTQ	*NA*	*NA*	*NA*
MPCA-E	0.054	0.149	0.345
KSSQ	**0.078**	0.191	0.437
SOBE	0.064	0.189	0.403
JKM	0.077	**0.213**	**0.511**
CompQ	0.072	0.200	0.504

TEST RESULTS FOR SIFT1M, 64-BIT CODES

	recall@1	recall@10	recall@100
PQ	0.224	0.599	0.924
TC	0.205	0.535	0.877
RVQ	0.257	0.659	0.952
CKM/OPQ	0.243	0.638	0.940
APQ	0.298	0.741	**0.972**
CQ	0.288	0.716	0.967
E-LOPQ	0.297	0.703	0.957
OCKM	0.274	0.680	0.945
ERVQ	0.276	0.694	0.962
OTQ	0.317	0.748	0.972
MPCA-E	0.286	0.710	0.923
KSSQ	0.325	0.754	0.976
SOBE	0.282	0.701	0.962
JKM	0.323	0.759	0.980
CompQ	**0.352**	**0.795**	**0.987**

TEST RESULTS FOR GIST1M, 64-BIT CODES

	recall@1	recall@10	recall@100
PQ	0.076	0.218	0.504
TC	0.096	0.223	0.547
RVQ	0.113	0.325	0.676
CKM/OPQ	0.118	0.334	0.715
AQ/APQ	*NA*	*NA*	*NA*
CQ	0.135	0.377	0.729
E-LOPQ	0.116	0.331	0.656
OCKM	0.130	0.358	0.720
ERVQ	0.115	0.341	0.711
OTQ	*NA*	*NA*	*NA*
MPCA-E	0.110	0.312	0.662
KSSQ	0.136	0.396	0.741
SOBE	0.136	0.360	0.705
JKM	0.140	0.413	0.769
CompQ	**0.155**	**0.419**	**0.801**

As shown in Table 16, SOBE produces promising results as it outperforms all the prior work in GIST1M for 64-bit codes. In general SOBE is better than methods such as TC, PQ, OPQ/CKM, OCKM and ERVQ. JKM [P4] takes this one-step further and with the proposed optimizations, JKM leaves all the prior work behind in quantization performance as displayed in Table 16. Finally, CompQ [P5] is the last and the best method for 64-bit codes. It is only outperformed by KSSQ and JKM in 32-bit experiments.

Table 17: Comparison of Proposed Methods with Prior Work in Cost of Encoding

Method	Cost of Encoding	Cost of Encoding for Different Datasets and Code Lengths (Number of Operations)			
		SIFT1M-32	**SIFT1M-64**	**GIST1M-32**	**GIST1M-64**
PQ	$O(KD)$	32768	32768	245760	245760
TC	$O(DL + \mathcal{B})$	**2592**	**5184**	**19232**	**38464**
RVQ	$O(MKD)$	131072	262144	983040	1966080
CKM/OPQ	$O(D^2 + KD)$	49152	49152	1167360	1167360
AQ	$O(M^2K^2(M + log(MK)) + KD)$	14712832	79724544	14925824	79937536
APQ	$O\left(D^2 + \frac{M}{4}(4^2K^2(4 + log(4K)) + KD)\right)$	14729216	14761984	15847424	16093184
CQ	$O(3MKD)$	393216	786432	2949120	5898240
E-LOPQ	$O(KD + KD + D^2)$	81920	81920	1413120	1413120
OCKM	$O(10KD)$	327680	327680	2457600	2457600
ERVQ	$O(MKD)$	131072	262144	983040	1966080
OTQ	$O(D^2 + KD + MK^2)$	311296	573440	1429504	1691648
MPCA-E	$O(2\mathcal{K}DL + \mathcal{K}\mathcal{B})$	1318912	2637824	1229824	2459648
KSSQ	$O(\mathcal{K}D + 2\kappa DL + \kappa\mathcal{B})$	115200	197632	338176	645632
SOBE	$O(2MKD)$	262144	524288	1966080	3932160
JKM	$O\left(MDK + \frac{(M-1)(M-2)}{2}KH + MKH\ log\ H\right)$	319488	761856	1171456	2465792
CompQ	$O\left(MDK + \frac{(M-1)(M-2)}{2}KH + MKH\ log\ H\right)$	319488	761856	1171456	2465792
K: number of sub-codewords		256	256	256	256
$\mathcal{K}$: number of subspaces (KSSQ/MPCA-E)		256	256	32	32
κ: number of candidate quantizers (KSSQ)		16	16	8	8
D: number of dimensions		128	128	960	960
M: number of sub-codebooks		4	8	4	8
L: number of reduced dimensions		20	40	20	40
$\mathcal{B}$: number of bits for encoding		32	64	32	64
H: number of candidates (JKM/CompQ)		32	32	32	32

In terms of computational cost of encoding, TC is still the fastest, but the proposed methods also provide comparable costs to the prior work. As it can be seen in Table 17, the improvements proposed for KSSQ decreases the computational cost significantly compared to MPCA-E. Among the proposed methods, KSSQ is the fastest. It is also faster than the best method in prior work, OTQ, but still produces better results. JKM and CompQ provide significant increase in performance compared to OTQ, while they have comparable computational costs.

Table 18: Comparison of Proposed Methods with Prior Work in Storage Cost

Method	Cost of Encoding	Storage Cost for Different Datasets and Code Lengths (MB)			
		SIFT1M-32	SIFT1M-64	GIST1M-32	GIST1M-64
PQ	$O(KD)$	0.25	0.25	1.88	1.88
TC	$O(DL)$	**0.02**	**0.04**	**0.15**	**0.29**
RVQ	$O(MKD)$	1.00	2.00	7.5	15
CKM/OPQ	$O(D^2 + KD)$	0.38	0.38	8.91	8.91
AQ	$O(MKD)$	1.00	2.00	7.5	15
APQ	$O(D^2 + MKD)$	1.13	2.13	14.53	22.03
CQ	$O(MKD)$	1.00	2.00	7.5	15
E-LOPQ	$O\big(K(KD + D^2)\big)$	96.00	96.00	2280.00	2280.00
OCKM	$O(D^2 + 2KD)$	0.63	0.63	10.78	10.78
ERVQ	$O(MKD)$	1.00	2.00	7.5	15
OTQ	$O(D^2 + MKD)$	1.13	2.13	14.53	22.03
MPCA-E	$O(\mathcal{K}DL)$	5.00	10.00	4.69	9.38
KSSQ	$O(\mathcal{K}DL)$	5.00	10.00	4.69	9.38
SOBE	$O(MKD)$	1.00	2.00	7.5	15
JKM	$O(MKD)$	1.00	2.00	7.5	15
CompQ	$O(MKD)$	1.00	2.00	7.5	15
$\boldsymbol{K}$: number of sub-codewords		256	256	256	256
$\mathcal{K}$: number of subspaces		256	256	32	32
$\boldsymbol{D}$: number of dimensions		128	128	960	960
$\boldsymbol{M}$: number of sub-codebooks		4	8	4	8
$\boldsymbol{L}$: number of reduced dimensions (average for KSSQ)		20	40	20	40

In terms of additional storage requirements, the proposed methods are comparable with each other and the methods in the literature. Multiple subspace based methods MPCA-E and KSSQ require more space compared to the other proposed methods for SIFT1M, but this is not the case for GIST1M. SOBE, JKM and CompQ share the same storage cost with methods like AQ and CQ, and even less cost than the best method in the prior art, OTQ.

4.4 A Vector Quantization Based K-NN Approach for Large-Scale Image Classification

The last contribution of this thesis to the VQ for ANN is a demonstration showing that VQ-based ANN is an efficient replacement for NN in other problems, such as classification on large-scale datasets [P6]. In [P6], it is proposed to develop a k-NN classifier, which uses VQ-based distance approximations and compare the obtained results with

the traditional k-NN. The Support Vector Machines (SVM) [136], [137] is used as a baseline.

The classification problem can be described as follows: Given a test sample $\boldsymbol{q} \in \mathbb{R}^D$, and a set of N reference samples $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\} \in \mathbb{R}^{D \times N}$, a label set $\boldsymbol{L} = \{l_1, l_2, \dots, l_N\} \in \mathbb{R}^{1 \times N}$ contains the labels of sample vectors. The label $l_i \in \{1, 2, \dots C\}$ is an integer between 1 and the number of classes C. A classifier $\mathbb{C}$ uses the sample set $\boldsymbol{X}$ and the label set $\boldsymbol{L}$ to map a given input $\boldsymbol{q}$ to its corresponding label, i.e., $\mathbb{C}(\boldsymbol{q}) = l_{\boldsymbol{q}}$.

1-NN classifier $\mathbb{C}_{1NN}$ maps a given input $\boldsymbol{q}$ to its corresponding label using the sorted reference set $\overline{\boldsymbol{X}}$, where $d(\boldsymbol{q}, \overline{\boldsymbol{x}}_i) \leq d(\boldsymbol{q}, \overline{\boldsymbol{x}}_j) \Leftrightarrow i < j$, i.e., $\mathbb{C}_{1NN}(\boldsymbol{q}) = l_1$. And this can be generalized to k-NN in a majority voting sense, which can be formulated as follows:

$$\mathbb{C}_{k-NN}(\boldsymbol{q}) = \underset{c}{\operatorname{argmax}} \sum_{k=1}^{K} \Theta(c = l_k) \quad \Theta(a) = \begin{cases} 1 & a = true \\ 0 & otherwise \end{cases}. \tag{4.22}$$

In other words, a k-NN classifier finds the k nearest neighbors to a given sample and performs majority voting[26] among the labels of the nearest neighbors. The class label with the greatest number of votes is returned as the classification result [138].

k-NN's greatest advantage compared to the other classification methods is that, it does not require learning. However, this is also a disadvantage because k-NN needs to store the reference sample set and perform NN for each given input [138]. This is costly in terms of both computations and storage. In [P6], the adaptation of VQ-based ANN to k-NN classifiers is proposed in order to overcome these drawbacks. VQ-based ANN approximates the distances while accelerating the distance calculation and compressing the dataset, which solves the two major problems of k-NN classifiers.

[P6] compares the performance of VQ-based k-NN to traditional k-NN and SVM in three datasets [139]. The *"PascalVOC07"* and *"ImageNet"* datasets are used with *"decaf7"* features, and the *"Bing"* dataset is used with "*decaf6*" features. The "*decaf*" features are convolutional neural network based features [140], extracted from the 6th (*decaf6*) and

[26] Weighted voting schemes have also been proposed in the literature [141]–[145], and their performances have been investigated in [P5].

the 7th (*decaf6*) layers of the network. Three quarters of each dataset is used as reference samples and the rest is used as the holdout test set. The properties of the datasets are presented in Table 19.

Table 19: Properties of Datasets Tested in [P5]

Property	Pascal	Bing	ImageNet
Number of Classes	20	257	118
Number of Training Samples	10740	90692	124200
Number of Test Samples	3580	30230	41400
Number of Dimension	4096	4096	4096
Feature Type	decaf7	decaf6	decaf7

VQ adaptation of k-NN is performed by using an RVQ quantizer with 128 bits (16 layers, 256 codevectors per layer). The non-exhaustive search property of RVQ is also used to accelerate the computations. Accuracy is used as the performance metric, which is the ratio of the number of correctly classified samples to the total number of samples. The obtained results are presented in Table 20.

Table 20: Classification Accuracies Obtained on Different Datasets
128-bit, K=64

Method	Pascal	Bing	ImageNet
k-NN	0.520	0.255	0.645
VQ k-NN	**0.532**	0.241	0.647
SVM	0.363	**0.260**	**0.701**

Table 20 shows that VQ-based k-NN provides comparable results with traditional k-NN. k-NN based methods outperform SVM notably in the Pascal dataset. In the Bing dataset, the obtained accuracies are comparable, while in ImageNet, SVM outperforms k-NN based methods. The computational costs and storage requirements of the aforementioned methods are presented in Table 21.

Table 21: Computational Costs and Storage Requirements

Method	**Computational Cost**	**Pascal**	**Bing**	**ImageNet**
k-NN	$O(ND + N\log_2 k)$	44055480	372018584	509468400
VQ k-NN	$O\left(\left(\frac{M^2}{2}+M\right)\frac{N}{e}+\frac{N}{e}\log_2 k + KMD\right)$	16877904	**17627454**	**17941591**
SVM	$O\left(\left(\frac{\varsigma^2-\varsigma}{2}\right)D\right)$	**778240**	134742016	28274688
Method	**Storage Requirement**	**Pascal**	**Bing**	**ImageNet**
k-NN	$O(ND)$	335MB	2834MB	3881MB
VQ k-NN	$O(N\mathcal{B} + KMD)$	128MB	**129MB**	**130MB**
SVM	$O\left(\left(\frac{\varsigma^2-\varsigma}{2}\right)D\right)$	**6MB**	1028MB	216MB
Parameters		**Pascal**	**Bing**	**ImageNet**
$\boldsymbol{\varsigma}$: Number of Classes		20	257	118
$\boldsymbol{N}$: Number of Samples		10740	90692	124200
$\boldsymbol{D}$: Number of Dimensions		4096	4096	4096
$\boldsymbol{M}$: Number of RVQ Layers		16	16	16
$\boldsymbol{K}$: Number of RVQ Codevectors		256	256	256
$\boldsymbol{\mathcal{B}}$: RVQ Codelength (bytes)		16	16	16
$\boldsymbol{e}$: Non-Exhaustive Search Constant		16	16	16
$\boldsymbol{k}$: k-Nearest Neighbours		64	64	64

For a given novel input sample, VQ k-NN first initializes the RVQ, which requires $O(KMD)$ computations. Then RVQ method requires $M^2/2 + M$ look-ups for distance approximations and using the non-exhaustive search, only N/e approximations are calculated, resulting in a computational cost of $O\left(\left(\frac{M^2}{2}+M\right)\frac{N}{e}+KMD\right)$. Compared to traditional k-NN's cost of $O(ND)$, this provides a significant improvement when N is large. Sorting the best k among N/e takes $O\left(\frac{N}{e}\log_2 k\right)$ operations. The required number of operations for SVM to predict the class of a given sample is $O\left(\left(\frac{\varsigma^2-\varsigma}{2}\right)D\right)$ [137].

SVM's storage requirement depends on the number of classes, so in Bing dataset, where $\varsigma = 257$, SVM requires almost 8 times more space than VQ-based k-NN. However, in Pascal, where $\varsigma = 20$, VQ-based k-NN requires 20 times more space than SVM. While the traditional k-NN requires storing the whole reference set, RVQ stores a compressed version of it. Although the difference is not that large in Pascal dataset where $N = 10740$, in ImageNet, where $N = 124200$, traditional k-NN requires 20 times more space than its VQ-based version. With these results, it is shown that VQ-based k-NN is an efficient approximation for traditional k-NN, especially in large datasets.

5 Conclusions

This thesis aims to propose novel solutions for one of the most important problems of our era, the similarity search problem on very large-scale datasets, which suffers from the exponential growth in data size and dimensions drastically. Thanks to the Vector Quantization based methods proposed in this thesis, it is possible to compress very large-scale datasets in extreme levels. In addition, this approach provides fast distance approximations. These two properties of the proposed methods target the Big Data problem, hence, this thesis is a direct effort to improve the efficiency of today's similarity search algorithms so that they will be able to handle Big Data. The methods proposed in this thesis focus on a subset of similarity search problems (Euclidean similarity) but it is possible to extend these methods to other distance metrics or similarity measures.

The first method proposed in this thesis [P1] aims to take advantage of subspace clustering techniques and improve the quantization performance. Many methods in the literature limited themselves to only one space, which generated inferior results. It is shown in the detailed experiments provided in [P1] that increasing the number of subspaces improves the quantization performance. However, this operation comes with a heavy computational cost, which needs further improvements.

The second algorithm [P2] improves the computational requirements of [P1], proving that the idea of performing quantization in multiple subspaces is applicable. [P2] also proposes a well-tailored optimization scheme for the proposed solution, which engages the optimization of subspace generation with optimization of quantizers. The proposed scheme performs better than the prior work, providing state-of-the-art approximate nearest neighbor retrieval results.

In [P3], the popular data visualization and clustering algorithm Self-Organizing Maps is introduced to the approximate nearest neighbor solution. The structure proposed in self-organizing maps is coupled with another hierarchically structured method, Residual Vector Quantization, in order to provide better search performance. The results presented in [P3] prove that the proposed structure enhances the quantization quality, by showing an improvement in the nearest neighbor approximation.

Inspired by the success of [P3], [P4] proposes hierarchical schemes to improve the quantization quality, in order to achieve better nearest neighbor approximations. Generalizing the popular K-Means clustering method to multiple hierarchical layers, [P4] scales it up to very large-scale datasets, enabling very efficient and successful clustering. The results

obtained in [P4] also proves that the proposed method outperforms the prior work for the approximate nearest neighbor search problem.

The last method in this thesis [P5] provides a very detailed analysis of the vector quantization problem and the distance approximation approach based on vector quantization. Using this analysis, [P5] proposes a solution based on stochastic gradient descent optimization, which is a well-proven optimization method. With this approach, state-of-the-art results are achieved in terms of retrieval accuracy, keeping the computational costs and additional storage requirements at a comparable level.

Finally, [P6] presents a demonstration of the applicability of the aforementioned solutions to the classification problem, introducing a vector quantization based k-Nearest Neighbor classifier. The method uses approximate distances to find the nearest neighbors to a given input and using majority-voting, the classification of the input is performed. The results provided in [P6] show that the proposed method is more efficient than traditional k-NN and SVM on very large-scale datasets with a high number of classes.

To conclude, several methods have been proposed in this thesis to solve the approximate nearest neighbor search problem, which is gaining higher and higher importance with today's ever-growing data. The proposed approaches achieves significantly better results than the prior art, proving the validity of the introduced concepts. Adaptation of VQ-based ANN to classification problems on large-scale datasets is shown as an applicable example of the benefits of the proposed ideas in this thesis. Even though the focus is kept on the approximation of the Euclidean distance, in the future it is also aimed to extend these approaches to other similarity measures such as cosine similarity, Manhattan distance or even Minkowski distance, which is the generic case of the Euclidean distance.

References

[1] R. M. Gray and D. L. Neuhoff, “Quantization,” *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[2] P. Gautreau, T. Ragab, C. Yanbiao, and C. Basaran, “Phonon dispersion and quantization tuning of strained carbon nanotubes for flexible electronics,” *Journal of Applied Physics*, vol. 115, no. 24, 2014.

[3] T. Kodama, K. Morita, G. Cincotti, and K. Kitayama, “A low-power photonic quantization approach using OFDM subcarrier spectral shifts,” *Optics express*, vol. 22, no. 23, pp. 28719–28730, 2014.

[4] J. P. Isaacs and J. C. Lamb, “Complementarity in biology; quantization of molecular motion,” Johns Hopkins Press, 1969.

[5] J. E. Avron, R. Seiler, and B. Simon, “Homotopy and quantization in condensed matter physics,” *Physical review letters*, vol. 51, no. 1, p. 51, 1983.

[6] P. Jørgensen, *Second quantization-based methods in quantum chemistry*. Elsevier, 2012.

[7] S. Lloyd, “Least squares quantization in PCM,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[8] W. F. Sheppard, “On the calculation of the most probable values of frequency-constants, for data arranged according to equidistant division of a scale,” *Proceedings of the London Mathematical Society*, vol. 1, no. 1, pp. 353–380, 1897.

[9] A. Reeves, “Pulse coded modulation,” French Patent, 852, 1938.

[10] B. M. Oliver, J. R. Pierce, and C. E. Shannon, “The philosophy of PCM,” *Proceedings of the IRE*, vol. 36, no. 11, pp. 1324–1331, 1948.

[11] W. R. Bennett, “Spectra of quantized signals,” *Bell Labs Technical Journal*, vol. 27, no. 3, pp. 446–472, 1948.

[12] L. G. Roberts, “Picture coding using pseudo-random noise,” *IRE Transactions on Information Theory*, vol. 8, no. 2, pp. 145–154, 1962.

[13] A. K. Jain, “Data clustering: 50 years beyond K-means,” *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[14] N. Jayant, “Digital coding of speech waveforms: PCM, DPCM, and DM quantizers,” *Proceedings of the IEEE*, vol. 62, no. 5, pp. 611–632, 1974.

[15] H. Kramer and M. Mathews, “A linear coding for transmitting a set of correlated signals,” *IRE Transactions on Information Theory*, vol. 2, no. 3, pp. 41–46, 1956.

[16] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Pearson, 2007.

[17] C. E. Shannon, “A mathematical theory of communication,” *Bell Labs Technical Journal*, vol. 27, pp. 379–423, 1948.

[18] D. A. Huffman, “A method for the construction of minimum redundancy codes,” *Proceedings of IRE*, vol. 40, pp. 1098–1101, 1952.

[19] G. K. Wallace, “The JPEG still picture compression standard,” *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[20] D. Slepian, “Permutation modulation,” *Proceedings of the IEEE*, vol. 53, no. 3, pp. 228–236, 1965.

[21] T. Berger, “Minimum entropy quantizers and permutation codes,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 149–157, 1982.

[22] T. Berger, F. Jelinek, and J. K. Wolf, “Permutation Codes for Sources,” *IEEE Transactions on Information Theory*, vol. 18, no. 1, pp. 160–169, 1972.

[23] H. Steinhaus, “Sur la division des corps materiels en parties,” *Bulletin of the Polish Academy of Sciences*, vol. 4, no. 3, pp. 801–804, 1956.

[24] D. L. Chaffee, “Applications of rate distortion theory to the bandwidth compression of speech signals,” University of California, Los Angeles, 1975.

[25] E. E. Hilbert, “Cluster compression algorithm: a joint clustering/data compression concept.,” *Jet Propulsion Lab*, vol. 77, no. 43, 1977.

[26] D. Chen, “On two or more dimensional optimum quantizers,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1977.

[27] A. Gersho, “Asymptotically optimal block quantization,” *IEEE Transactions on Information Theory*, vol. 25, no. 4, pp. 373–380, 1979.

[28] W.-Y. Chan and A. Gersho, “Generalized product code vector quantization: A family of efficient techniques for signal compression,” *Digital Signal Processing*, vol. 4, no. 2, pp. 95–126, 1994.

[29] M. L. Sabin and R. Gray, “Product code vector quantizers for waveform and voice coding,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 3, pp. 474–488, 1984.

[30] T. R. Fischer, “A pyramid vector quantizer,” *IEEE Transactions on Information Theory*, vol. 32, no. 4, pp. 568–583, 1986.

[31] D. F. Lyons and D. L. Neuhoff, “A coding theorem for low-rate transform codes.,” in *IEEE International Symposium on Information Theory*, 1993.

[32] M. Vetterli and J. Kovačević, *Wavelets and subband coding*. Prentice-Hall, 2007.

[33] A. Buzo, A. Gray, R. M. Gray, and J. D. Markel, “Speech coding based upon vector quantization,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 5, pp. 562–574, 1980.

[34] J. Biing-Hwang and A. Gray, “Multiple stage vector quantization for speech coding,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1982, vol. 7, no. 3, pp. 597–600.

[35] A. D. Subramaniam and B. D. Rao, “PDF optimized parametric vector quantization of speech line spectral frequencies,” in *IEEE Workshop on Speech Coding: Meeting the Challenges of the New Millennium*, 2000, vol. 11, no. 2, pp. 87–89.

[36] C.-C. Chang, W.-L. Tai, and C.-C. Lin, “A reversible data hiding scheme based on side match vector quantization,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 10, pp. 1301–1308, 2006.

[37] J. S. Pan, Z. M. Lu, and S. H. Sun, “An efficient encoding algorithm for vector quantization based on subvector technique,” *IEEE Transactions on Image Processing*, vol. 12, no. 3, pp. 265–270, 2003.

[38] M. Fleming, Q. Zhao, and M. Effros, “Network vector quantization,” *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1584–1604, 2004.

[39] S. Dasgupta and Y. Freund, “Random projection trees for vector quantization,” *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3229–3242, 2009.

[40] N. B. Karayiannis, “Soft learning vector quantization and clustering algorithms based on ordered weighted aggregation operators,” *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1093–1105, 2000.

[41] V. A. Vaishampayan, N. J. A. Sloane, and S. D. Servetto, “Multiple-description vector quantization with lattice codebooks: Design and analysis,” *IEEE Transactions on Information Theory*, vol. 47, no. 5, pp. 1718–1734, 2001.

[42] A. Vasuki and P. T. Vanathi, “A review of vector quantization techniques,” *IEEE Potentials*, vol. 25, no. 4, pp. 39–47, 2006.

[43] P. Indyk and R. Motwani, “Approximate nearest neighbors: Towards removing the curse of dimensionality,” in *ACM Symposium on Theory of Computing*, 1998, pp. 604–613.

[44] K. Q. Weinberger and K. S. Lawrence, “Distance metric learning for large margin nearest neighbor classification,” *Journal of Machine Learning Research*, vol. 10, no. May, pp. 207–244, 2009.

[45] M. de Berg, O. Cheong, M. van Krevald, and M. Overmars, *Computational geometry: algorithms and applications*, 3rd ed. Berlin, Heidelberg: Springer-Verlag, 2008.

[46] R. Szeliski, *Computer vision: algorithms and Applications*, 1st ed. London: Springer-Verlag, 2010.

[47] G. Shakhnarovich, T. Darrell, and P. Indyk, *Nearest-neighbor methods in learning and vision: theory and practice*. The MIT Press, 2006.

[48] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[49] O. Boiman, E. Shechtman, and M. Irani, “In defense of nearest-neighbor based image classification,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008, no. i, pp. 1–8.

[50] X. Wu, V. Kumar, Q. J. Ross, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z. H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, “Top 10 algorithms in data mining,” *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2008.

[51] A. Dhurandhar and A. Dobra, “Probabilistic characterization of nearest neighbor classifier,” *International Journal of Machine Learning and Cybernetics*, vol. 4, no. 4, pp. 259–272, 2013.

[52] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is ‘nearest-neighbor’ meaningful?,” in *International Conference on Database Theory (ICDT)*, 1999, pp. 217–235.

[53] B. Siddiquie, R. Feris, and L. Davis, “Image ranking and retrieval based on multi-attribute queries,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 801–808.

[54] M. Davy and S. J. Godsill, *Audio information retrieval: A bibliographical study*. Citeseer, 2002.

[55] M. Lew, N. Sebe, C. Djeraba, and R. Jain, “Content-based multimedia information retrieval: State of the art and challenges,” *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, vol. 2, no. 1, pp. 1–19, 2006.

[56] D. W. Aha, D. Kibler, and M. K. Albert, “Instance-based learning algorithms,” *Machine Learning*, vol. 6, no. 1, pp. 37–66, 1991.

[57] D. P. Dobkin and R. J. Lipton, “Multidimensional searching problems,” *SIAM Journal on Computing*, vol. 5, no. 2, pp. 181–186, 1976.

[58] J. Matoušek, “Reporting points in halfspaces,” *Computational Geometry: Theory and Applications*, vol. 2, no. 3, pp. 169–186, 1992.

[59] P. K. Agarwal and J. Matousek, “Ray shooting and parametric search,” *SIAM J Computing*, vol. 22, no. 4, pp. 794–806, 1993.

[60] S. Har-Peled and N. Kumar, “Approximate nearest neighbor search for low dimensional queries,” *SIAM Journal on Computing*, vol. 42, no. 1, pp. 138–159, 2011.

[61] N. Bhatia and V. Ashev, “Survey of nearest-neighbor techniques,” *International Journal of Computer Science and Information Security*, vol. 8, no. 2, pp. 302–305, 2010.

[62] T. Mei, Y. Rui, S. Li, and Q. Tian, “Multimedia search reranking,” *ACM Computing Surveys*, vol. 46, no. 3, pp. 1–38, 2014.

[63] J. Wang, H. T. Shen, J. Song, and J. Ji, “Hashing for similarity search: a survey,” in *arXiv preprint*, 2014, p. :1408.2927.

[64] J. Wang, W. Liu, S. Kumar, and S.-F. Chang, “Learning to hash for indexing big bata-a survey,” *Proceedings of the IEEE*, vol. 104, no. 1, pp. 34–57, Jan. 2016.

[65] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, “A Survey on Learning to Hash,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 9, 2017.

[66] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, “An optimal algorithm for approximate nearest neighbor searching in fixed dimensions,” *Journal of the ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[67] J. S. Beis and D. G. Lowe, “Shape indexing using approximate nearest-neighbour search in high-dimensional spaces,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1997, pp. 1000–1006.

[68] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[69] J. H. Freidman, J. L. Bentley, and R. A. Finkel, “An Algorithm for Finding Best Matches in Logarithmic Expected Time,” *ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209–226, 1977.

[70] S. Arya, “Algorithms for fast vector quantization,” in *Data Compression Conference, 1993.*, 1993, pp. 1–17.

[71] R. F. Sproull, “Refinements to nearest-neighbor searching in k-dimensional trees,” *Algorithmica*, vol. 6, no. 1–6, pp. 579–589, 1991.

[72] I. T. Jolliffe, *Principal Component Analysis.* John Wiley & Sons, 2002.

[73] S. Dasgupta and Y. Freund, “Random projection trees and low dimensional manifolds,” in *ACM Symposium on Theory of Computing*, 2008, vol. 6, no. 1, p. 537.

[74] T. Liu, A. W. Moore, A. Gray, and K. Yang, “An investigation of practical approximate nearest neighbor algorithms,” *Advances in Neural Information Processing Systemsnformation*, p. 8, 2004.

[75] D. Nister and H. Stewenius, “Scalable recognition with a vocabulary tree,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006, vol. 2, pp. 2161–2168.

[76] C. Silpa-Anan and R. Hartley, “Optimised KD -trees for fast image descriptor matching,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008, pp. 1–8.

[77] M. Muja and D. G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” in *International Conference on Computer Vision Theory and Applications (VISAPP)*, 2009, vol. 2, no. 331–340, pp. 1–10.

[78] J. Wang, N. Wang, Y. Jia, J. Li, G. Zeng, H. Zha, and X.-S. Hua, “Trinary-Projection Trees for Approximate Nearest Neighbor Search,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 2, pp. 388–403, 2014.

[79] R. F. Lyon, M. Rehn, S. Bengio, T. C. Walters, and G. Chechik, “Sound retrieval and ranking using sparse auditory representations,” *Neural Computation*, vol. 22, no. 9, pp. 2390–416, Sep. 2010.

[80] D. Knuth, *The art of computer programming*. Addison-Wesley, 1997.

[81] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, “Locality-sensitive hashing scheme based on p-stable distributions,” in *ACM Symposium on Computational Geometry*, 2004, p. 253.

[82] K. Terasawa and Y. Tanaka, “Spherical LSH for approximate nearest-neighbor search on unit hypersphere,” in *Workshop on Algorithms and Data Structures*, 2007, pp. 27–38.

[83] P. Indyk and A. Andoni, “Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions,” in *Foundations of Computer Science*, 2006, vol. 51, no. 1, pp. 117–122.

[84] David Gorisse and Matthieu Cord, “Locality-sensitive hashing for chi2 distance,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 10, pp. 2058–64, Oct. 2012.

[85] B. Kulis, P. Jain, and K. Grauman, “Fast similarity search for learned metrics,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2143–2157, 2009.

[86] Y. Weiss, A. Torralba, and R. Fergus, “Spectral hashing,” in *Advances in Neural Information Processing Systems (NIPS)*, 2009, pp. 1753–1760.

[87] J. Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. March, pp. 888–905, 2005.

[88] D. Zhang, J. Wang, D. Cai, and J. Lu, “Self-taught hashing for fast similarity search,” in *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010, pp. 18–25.

[89] L. Hagen and A. B. Kahng, “New spectral methods for ratio cut partitioning and clustering,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits*

and Systems, vol. 11, no. 9, pp. 1074–1085, 1992.

[90] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, “Hashing with graphs,” in *International Conference on Machine Learning (ICML)*, 2011, pp. 1–8.

[91] Y. Matsushita, “Principal component hashing: An accelerated approximate nearest neighbor search,” *Advances in Image and Video Technology*, pp. 374–385, 2009.

[92] H. Jegou, M. Douze, C. Schmid, and P. Perez, “Aggregating local descriptors into a compact image representation,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 3304–3311.

[93] Y. Gong and S. Lazebnik, “Iterative quantization: A procrustean approach to learning binary codes,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 817–824.

[94] W.-H. Kong and W.-J. Li, “Double-bit quantization for hashing,” in *AAAI: Conference on Artificial Intelligence*, 2012, pp. 634–640.

[95] W. Kong and W.-J. Li, “Isotropic hashing,” in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1–9.

[96] W. Kong, W. Li, and M. Guo, “Manhattan hashing for large-scale image retrieval categories and subject descriptors,” in *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2012, pp. 45–54.

[97] L. Zhang, Y. Zhang, J. Tang, K. Lu, and Q. Tian, “Binary code ranking with weighted hamming distance,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, no. 6, pp. 1586–1593.

[98] Y. G. Jiang, J. Wang, and S. F. Chang, “Lost in binarization: query-adaptive ranking for similar image search with compact codes,” in *ACM International Conference on Multimedia Retrieval*, 2011, p. 16.

[99] A. Gordo, F. Perronnin, Y. Gong, and S. Lazebnik, “Asymmetric distances for binary embeddings.,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 1, pp. 33–47, Jan. 2014.

[100] C. Wu, J. Zhu, D. Cai, C. Chen, and J. Bu, “Semi-supervised nonlinear hashing using bootstrap sequential projection learning,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 6, pp. 1380–1393, 2013.

[101] W. Liu, J. Wang, R. Ji, and Y. J. S. Chang, “Supervised hashing with kernels,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 2074–2081.

[102] T. Ge, K. He, and J. Sun, “Graph cuts for supervised binary coding,” in *European Conference on Computer Vision (ECCV)*, 2014, pp. 250–264.

[103] J. Wang, S. Kumar, and S.-F. Chang, “Semi-supervised hashing for large-scale

search.," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 12, pp. 2393–406, Dec. 2012.

[104] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for scalable image retrieval," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 3424–3431.

[105] J. Song, N. Sebe, D. Zhang, L. Gao, and Y. Yan, "Supervised Hashing with Pseudo Labels for Scalable Multimedia Retrieval," in *ACM international conference on Multimedia*, 2015, pp. 827–830.

[106] A. Dick, "Learning hash functions using column generation," in *International Conference on Machine Learning (ICML)*, 2013, vol. 28.

[107] M. Norouzi, D. J. D. D. J. Fleet, R. Salakhutdinov, and D. M. Blei, "Hamming distance metric learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1–9.

[108] J. Wang, J. Wang, N. Yu, and S. Li, "Order preserving hashing for approximate nearest neighbor search," in *ACM International Conference on Multimedia*, 2013, pp. 133–142.

[109] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou, "Deep hashing for compact binary codes learning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, vol. 07–12–June, pp. 2475–2483.

[110] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search.," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–28, Jan. 2011.

[111] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: re-rank with source coding," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, no. 3, pp. 861–864.

[112] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized Product Quantization.," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 4, pp. 744–755, Dec. 2014.

[113] Y. Chen, T. Guan, and C. Wang, "Approximate nearest neighbor search by residual vector quantization," *Sensors*, vol. 10, no. 12, pp. 11259–11273, 2010.

[114] H. Jegou, M. Douze, and C. Schmid, "Searching with quantization: approximate nearest neighbor search using short codes and distance estimators," INRIA, 2009.

[115] J. Brandt, "Transform coding for fast approximate nearest neighbor search in high dimensions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 1815–1822.

[116] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized Product Quantization for Approximate Nearest Neighbor Search," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, pp. 2946–2953.

[117] M. Norouzi and D. J. Fleet, “Cartesian K-Means,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, pp. 3017–3024.

[118] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[119] N. Kambhatla and T. K. Leen, “Dimension Reduction by Local Principal Component Analysis,” *Neural Computation*, vol. 9, no. 7, pp. 1493–1516, Oct. 1997.

[120] L. Ai, J. Yu, Z. Wu, Y. He, and T. Guan, “Optimized residual vector quantization for efficient approximate nearest neighbor search,” *Multimedia Systems*, Jun. 2015.

[121] A. Babenko and V. Lempitsky, “Additive quantization for extreme vector compression,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 931–938.

[122] T. Zhang, D. Chao, and J. Wang, “Composite Quantization for Approximate Nearest Neighbor Search,” in *International Conference on Machine Learning (ICML)*, 2014.

[123] Liefu Ai, Junqing Yu, T. Guan, and Yunfeng He, “Efficient approximate nearest neighbor search by optimized residual vector quantization,” in *International Workshop on Content-Based Multimedia Indexing (CBMI)*, 2014, pp. 1–4.

[124] S. Shapiro, “Beam Search,” *Encyclopedia of artificial intelligence*. 1987.

[125] Y. Kalantidis and Y. Avrithis, “Locally optimized product quantization for approximate nearest neighbor search,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[126] J. Wang, J. Wang, J. Song, X.-S. Xu, H. T. Shen, and S. Li, “Optimized cartesian k-means,” *IEEE Transactions on Knowledge & Data Engineering*, vol. 27, no. 1, pp. 180–192, Jan. 2015.

[127] A. Babenko and V. Lempitsky, “Tree quantization for large-scale similarity search and classification,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[128] A. Babenko and V. Lempitsky, “The inverted multi-index,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, vol. 14, no. 1–3, pp. 3069–3076.

[129] H. Jegou, M. Douze, and C. Schmid, “Hamming embedding and weak geometry consistency for large scale image search,” in *European Conference on Computer Vision (ECCV)*, 2008, no. October, pp. 304–317.

[130] A. Oliva and A. Torralba, “Modeling the shape of the scene: A holistic representation of the spatial envelope,” *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145–175, 2001.

[131] a Torralba, R. Fergus, and F. W., “80 million tiny images: a large dataset for non-parametric object and scene recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[132] P. Agarwal and N. Mustafa, “K-means projective clustering,” in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2004, pp. 155–165.

[133] V. Gassenbauer, J. Křivánek, K. Bouatouch, C. Bouville, and M. Ribardière, “Improving performance and accuracy of local PCA,” *Computer Graphics Forum*, vol. 30, no. 7, pp. 1903–1910, Sep. 2011.

[134] T. Kohonen, “The self-organizing map,” *Neurocomputing*, vol. 21, no. 1–3, pp. 1–6, 1998.

[135] J. Lampinen and T. Kostiainen, “Overtraining and model selection with the self-organizing map,” in *International Joint Conference on Neural Networks (IJCNN)*, 1999, vol. 3, pp. 1911–1915.

[136] C. J. C. Burges, “A tutorial on support vector machines for pattern recognition,” *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[137] C.-C. Chang and C.-J. Lin, “Libsvm,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1–27, 2011.

[138] P. Cunningham and S. J. Delany, “K-nearest neighbour classifiers,” 2007.

[139] T. Tommasi and T. Tuytelaars, “A testbed for cross-dataset analysis,” in *European Conference on Computer Vision (ECCV)*, 2015, vol. 8927, pp. 18–31.

[140] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, “Decaf: A deep convolutional activation feature for generic visual recognition,” in *International Conference on Machine Learning (ICML)*, 2014, pp. 647–655.

[141] J. Gou, T. Xiong, and Y. Kuang, “A novel weighted voting for K-nearest neighbor rule,” *Journal of Computers*, vol. 6, no. 5, pp. 833–840, 2011.

[142] J. Gou, “A new distance-weighted k-nearest neighbor classifier,” *Journal of Information & Computational Science*, vol. 6, no. June, pp. 1429–1436, 2012.

[143] E. Fix and J. L. Hodges, “Discriminatory analysis nonparametric discrimination: consistency properties,” *International Statistical Review / Revue Internationale de Statistique*, vol. 57, no. 3, p. 238, Dec. 1989.

[144] T.-L. Pao, Y.-T. Chen, J.-H. Yeh, Y.-M. Cheng, and Y.-Y. Lin, “A comparative study of different weighting schemes on KNN-based emotion recognition in Mandarin speech,” *Advanced Intelligent Computing Theories and Applications. With Aspects of Theoretical and Methodological Issues*, pp. 997–1005, 2007.

[145] S. A. Dudani, “The distance-weighted k-nearest-neighbor rule,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-6, no. 4, pp. 325–327,

1976.

ORIGINAL PAPERS

I

M-PCA BINARY EMBEDDING FOR APPROXIMATE NEAREST NEIGHBOR SEARCH

by

E.C.Ozan, S. Kiranyaz & M. Gabbouj, August 2015

IEEE Trustcom/BigDataSE/ISPA (BigDataSE), Helsinki, 2015.

M-PCA Binary Embedding For Approximate Nearest Neighbor Search

Ezgi Can Ozan Serkan Kiranyaz Moncef Gabbouj

Tampere University of Technology
Tampere, Finland
{name.surname}@tut.fi

***Abstract*—Principal Component Analysis (PCA) is widely used within binary embedding methods for approximate nearest neighbor search and has proven to have a significant effect on the performance. Current methods aim to represent the whole data using a single PCA however, considering the Gaussian distribution requirements of PCA, this representation is not appropriate. In this study we propose using Multiple PCA (M-PCA) transformations to represent the whole data and show that it increases the performance significantly compared to methods using a single PCA.**

Keywords—PCA, Binary Embedding, Hashing, Approximate Nearest Neighbor Search, Retrieval on Big Data

I. INTRODUCTION

With the recent progresses in the multimedia technology, there has been a tremendous increase in the amount of data produced by users, making multimedia data the "biggest big data". The "big" term for multimedia data is used regarding to its sheer size, and the amount of information it covers. Thanks to today's social media, almost everyone's daily experiences are carried onto the web, to be presented in an image, audio or video format. As the amount of data produced everyday becomes greater and greater, the problems of storage, indexing and retrieval are gaining more importance. As the increase in the dataset sizes creates major problems, researchers who have been working on this problem have focused on two major aspects. First, as the datasets become larger, they require more computational power for indexing and retrieval. Second, these datasets are represented by signatures with greater number of dimensions, which also creates a storage problem. In other words, as the number of items in a dataset and the dimensions of the descriptors increase, the memory required to process all items together at once also increases.

As a solution to this problem, binary code representations of descriptors are proposed. The aim is to represent a given descriptor in form of a much more compact binary string, such that the distances and similarities between original descriptor pairs are preserved. In other words, descriptors represented by similar binary strings are also similar. This solution directly targets both of the problems mentioned above. First, calculating the distance between binary codes requires much less computational cost then calculating the distance between vectors of double precision numbers. Second, transforming a descriptor into a binary string decreases the storage cost extremely. For example, a widely used image descriptor GIST[1] is usually represented by 512 double precision numbers. When this descriptor is represented by a 64-bit binary string, the memory footprint is reduced by 512 times. Also calculating the Euclidean distance between two such descriptors takes 512 subtractions, square and additions while Hamming distance is calculated by a logical operator, which brings a significant decrease for the required computational cost.

II. RELATED WORK

One of the earliest approaches for binary coded representations of descriptors is Locality Sensitive Hashing [2] (LSH). In LSH, Datar et al. propose projecting the data onto random vectors generated by a probabilistic model and perform binary embedding on these projections. LSH has been a successful method and further extensions on LSH have been proposed, as some important examples can be found in [3]–[5]. Another mile stone for binary embedding is Spectral Hashing [6] (SH). In SH, Weiss et al. propose a method based on spectral clustering. This method also has been very successful in the field of binary embedding, and several extensions have been proposed, as some examples can be found in [7]–[9].

A very simple but successful technique for hashing has been proposed by Gordo et al. in [10] named PCA Embedding (PCAE), where principal component analysis (PCA) is used to reduce dimension and generate the projection vectors for hashing. However, since the information is distributed non-uniformly between principal components, several methods have been proposed to balance the variance among the projections. Performing random rotation on PCA (PCA-RR) is shown to be a viable solution in [11] by Gong et al. Further in this study, an iterative approach to optimize the rotation has been proposed, named Iterative Quantization (ITQ). ITQ aims to rotate the data after PCA to balance the variance on each principal component. Another approach to balance the variance distribution has been proposed by Brandt in [12] named Transform Coding (TC). In TC, the bits are distributed non-uniformly among dimensions after PCA.

As presented above PCA transformation has become an important step for dimension reduction in binary embedding and

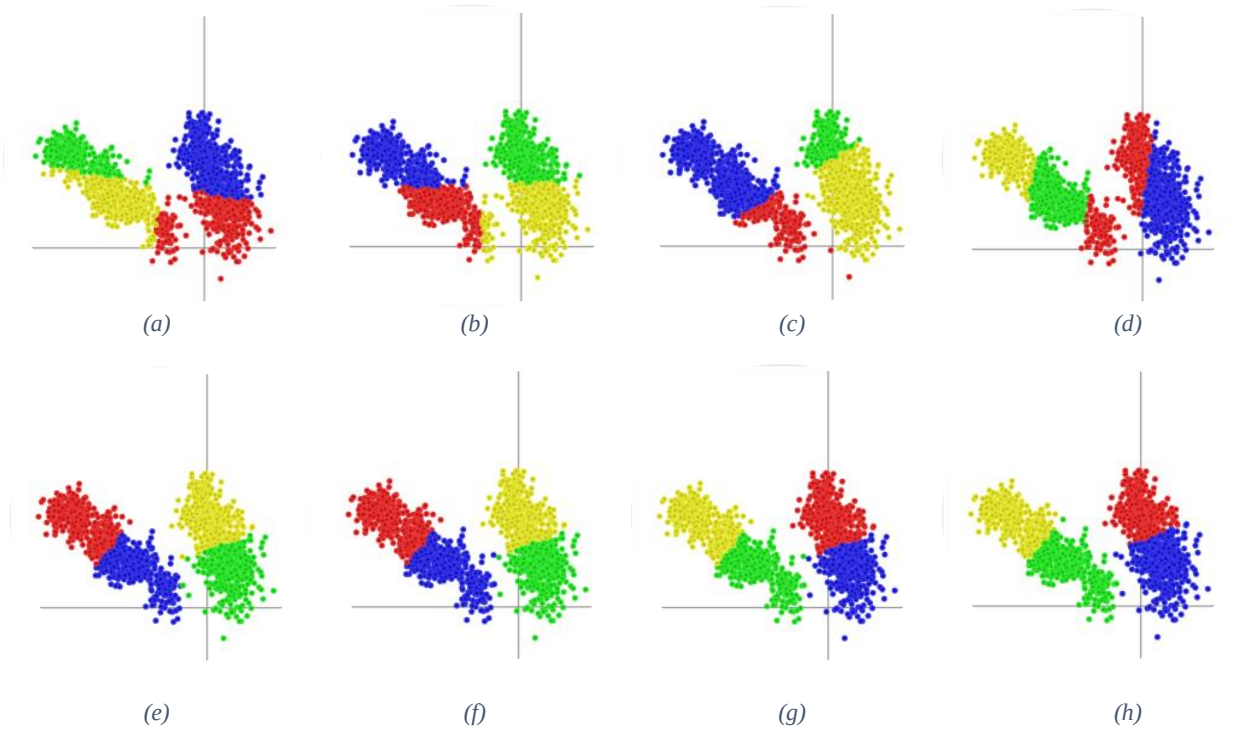

Figure 1: A toy example showing the effect of using multiple PCAs. First row are the results of methods PCAE (a), PCA-RR (b), ITQ (c) and TC (d) respectively. The second row is their M-PCA versions as M-PCAE (e), M-PCA-RR (f), M-PCA-ITQ (g) and M-PCA-TC (h). (M=2, colors only represent different hash codes)

hashing methods. Yet PCA comes with an assumption that, the data have Gaussian distribution properties, and this assumption is highly likely to fail when large datasets are taken into consideration. However, representing the data with multiple PCA's have been shown to be a better approximation of data, as in [13]–[16].

Here in this study, we propose that using multiple PCA's instead of a single PCA increases the performance of hashing systems using asymmetric distances. We propose M-PCA versions of traditional hashing methods based on PCA such as PCAE, PCA-RR, ITQ and TC and show that the performance is significantly increased. The rest of the paper is organized as follows: In the next section, the problem formulation is defined. In Section IV, the proposed method is described in detail. In Section V, experiments and obtained results are presented. Finally in Section VI, the paper is concluded.

III. Problem Formulation

In this section we present the problem formulation and further discuss the selected binary embedding methods based on PCA. The problem formulation for binary embedding can be given as follows: Given a set of N vectors $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$, $(\boldsymbol{x}_i \in \mathbb{R}^D)$, a set of K functions $H = \{h_1, \dots, h_K\}$ is defined to perform multidimensional binary embedding $h(\boldsymbol{x}_i) = [h_1(\boldsymbol{x}_i), \dots, h_K(\boldsymbol{x}_i)]^T$, i.e. $h_k: \mathbb{R} \to \{0,1\}$, where $h_k(x) \in \{0,1\}$, and $h(x) \in \{0,1\}^K$. Within the scope of this paper we can further decompose $h_k(\boldsymbol{x}_i)$ as given in Eq. (1)

$$h_k(\boldsymbol{x}_i) = s_k\big(e_k(\boldsymbol{x}_i)\big) \tag{1}$$

where e_k is an embedding function and s_k is a binarization function, such that $e_k(\boldsymbol{x}_i) \in \mathbb{R}$ and $s_k\big(e_k(\boldsymbol{x}_i)\big) \in \{0,1\}$. We select the embedding function e_k such that it preserves the Euclidean distance between two samples as presented in Eq. (2)

$$\|\boldsymbol{x}_i, \boldsymbol{x}_j\|^2 \cong \sum_k^K \|e_k(\boldsymbol{x}_i), e_k(\boldsymbol{x}_j)\|^2 \tag{2}$$

A. PCA-Embedding (PCAE)

As presented in [10], PCAE aims to perform PCA transformation on a training dataset then according to each projected dimension, binary codes are generated. Following the given descriptions above, PCAE can be formulated as given in Eq. (3)

$$\begin{aligned} e_k(\boldsymbol{x}_i) &= \boldsymbol{w}_k^T(\boldsymbol{x}_i - \boldsymbol{\mu}) \\ s_k(y) &= sign(y) \end{aligned} \tag{3}$$

where $\boldsymbol{W}$ is the PCA transformation matrix ($\boldsymbol{W} \in \mathbb{R}^{D \times K}$), $\boldsymbol{w}_k$ is the k[th] row of $\boldsymbol{W}$ and $\boldsymbol{\mu}$ is the sample mean of $\boldsymbol{X}$.

B. PCA + Random Rotations (PCA-RR)

As a result of the PCA dimension reduction, the dimensions are sorted with decreasing variance. A simple solution to imbalanced distribution of variances among dimensions after PCA transformation is presented in [11]. It is proposed that, the PCA transformation matrix $\boldsymbol{W}$ is multiplied by a random orthogonal rotation matrix $\boldsymbol{R}$ ($\boldsymbol{R} \in \mathbb{R}^{D \times D}$) in order to obtain a

more balanced distribution of variances. Note that since $\boldsymbol{R}$ is an orthogonal rotation, it preserves Euclidean distance. PCA-RR can be formulated as given in Eq. (4)

$$\begin{aligned} \widetilde{\boldsymbol{W}} &= \boldsymbol{R}\boldsymbol{W} \\ e_k(\boldsymbol{x}_i) &= \widetilde{\boldsymbol{w}}_k^T(\boldsymbol{x}_i - \boldsymbol{\mu}) \\ s_k(y) &= sign(y) \end{aligned} \tag{4}$$

C. Iterative Quantization (ITQ)

ITQ [11] takes the random rotation approach presented above one step further and using an iterative Procrustean approach it tries optimize the orthogonal rotation minimizing the quantization error. The ITQ can be simply formulated as given in Eq. (5)

$$\begin{aligned} \widetilde{\boldsymbol{W}} &= \boldsymbol{R}_{ITQ}\boldsymbol{W} \\ e_k(\boldsymbol{x}_i) &= \widetilde{\boldsymbol{w}}_k^T(\boldsymbol{x}_i - \boldsymbol{\mu}) \\ s_k(y) &= sign(y) \end{aligned} \tag{5}$$

D. Transform Coding (TC)

In TC [12], instead of performing additional rotations as the two methods given above, binary embedding is performed non-uniformly on each dimension after PCA, i.e. each dimension can be allocated different number of bits. The non-uniform distribution of bits provides equal distribution of variance among binary codes. The formulation of TC is given in Eq. (6)

$$\begin{aligned} e_k(\boldsymbol{x}_i) &= \boldsymbol{w}_k^T(\boldsymbol{x}_i - \boldsymbol{\mu}) \\ s_k(y) &= \underset{c \in C_k}{\operatorname{argmin}} \|y - c\| \end{aligned} \tag{6}$$

where C_k is the set of centroids on a given dimension m.

The methods are illustrated in a toy example, as shown in Figure 1. As it can be seen, using multiple PCA's improves the hashing quality.

IV. Proposed Method

As explained in the previous section, PCA is widely used as a binary embedding function and binary embedding is performed on the projected space. However one single PCA may not be sufficient to successfully represent the whole training set, as the Gaussian distribution requirements may not hold. In such cases, introducing more than one PCA transformation is proven to provide better representations [13]–[16].

In our proposed approach we partition the training set into *M* affine subspaces to introduce multiple PCA's. We then perform binary embedding on the obtained subspaces. We represent each affine subspace $\boldsymbol{\mathcal{F}}_m$ with an affine shift $\boldsymbol{\mu}_m$ and a transformation matrix $\boldsymbol{W}_m$. In the next subsection we explain the subspace clustering approach followed in our method.

A. M-PCA Clustering

In order to cluster the training set into *M* affine subspaces, we follow an iterative approach. To initialize the clusters, we first perform K-Means clustering. For each cluster obtained, we perform PCA for dimension reduction and obtain the initial affine subspaces. Then we update the clusters by calculating the distance of each sample to subspaces and selecting the nearest subspace. The distance of a sample $\boldsymbol{x}_i$ to a given affine subspace $\boldsymbol{\mathcal{F}}_m$, defined by the projection matrix $\boldsymbol{W}_m$ and affine shift vector $\boldsymbol{\mu}_m$ is calculated using the formula given in Eq. (7)

$$d(\boldsymbol{x}_i, \boldsymbol{\mathcal{F}}_m) = \left\| {\boldsymbol{W}_m}^{\perp^T}(\boldsymbol{x}_i - \boldsymbol{\mu}_m) \right\| \tag{7}$$

where ${\boldsymbol{W}_m}^{\perp}$ spans the orthogonal complement of range space of $\boldsymbol{W}_m$. For each updated cluster, $\boldsymbol{W}_m$ and $\boldsymbol{\mu}_m$ is calculated again and the iterations continue until convergence or the maximum number of iterations is reached. After subspaces are obtained, any PCA-based binary embedding method can be further applied.

B. Asymmetric Distance Calculation

In [10], Gordo et al. has shown that using asymmetric distances for binary embedding methods boosts the performance notably, so in our approach we also propose asymmetric distances. One requirement of asymmetric distances is that, in order to calculate the distances between a query and a set of codes, the query should not be coded. Then reconstructing the vectors from codes, asymmetric distances can be calculated.

Asymmetric distance calculation can be taken one step further by mapping binary codes to pre-trained centroid values. As also presented in [10], here we map every binary code to the centroid it corresponds to. We use the sample mean as the centroid value, i.e. for each dimension, we calculate the average of samples that correspond to same bit value and store it in a look-up table. In our proposed method, the asymmetric distance of a given query vector $\boldsymbol{x_i}$ to a given code $\boldsymbol{c}_{m,j}$ is calculated as given in Eq. (8), (9) and (10). Here $\hat{\boldsymbol{x}}_{i,m}$ is the projection of $\boldsymbol{x_i}$ onto the subspace $\boldsymbol{\mathcal{F}}_m$, and subscript *j* represents different codes in the given subspace.

$$d(\boldsymbol{x_i}, \boldsymbol{c}_{m,j}) = \sqrt{d(\boldsymbol{x_i}, \hat{\boldsymbol{x}}_{i,m})^2 + d(\hat{\boldsymbol{x}}_{i,m}, \boldsymbol{c}_{m,j})^2} \tag{8}$$

$$d(\hat{\boldsymbol{x}}_{i,m}, \boldsymbol{c}_{m,j}) = \left\| \boldsymbol{W}_m^T(\boldsymbol{x}_i - \boldsymbol{\mu}_m) - \boldsymbol{c}_{m,j} \right\| \tag{9}$$

$$d(\boldsymbol{x}_i, \hat{\boldsymbol{x}}_{i,m}) = \left\| (\boldsymbol{x}_i - \boldsymbol{\mu}_m) - \boldsymbol{W}_m^T \boldsymbol{W}_m (\boldsymbol{x}_i - \boldsymbol{\mu}_m) \right\| \tag{10}$$

Here note that, the distance of a query vector to the projection space has not been taken into account in any of the asymmetric distance calculations given in [10], since there is only one subspace it does not affect the nearest neighbor search. In our method, as there are more than one subspaces, we also take this distance into account.

C. Encoding of Vectors

In traditional binary embedding methods, encoding of a database vector is performed as given in the Eq. (1). Here we modify this in order to introduce our multiple subspace approach. Since we introduce several affine subspaces to our system, first we need to decide on which subspace to perform binary embedding on. As we also choose to use asymmetric distances, our selection of subspace also depends on the asymmetric distance calculation. In order to determine the most suitable subspace for projection, we first project the given vector onto each subspace and apply binary embedding there. Then for each alternative code we calculate the asymmetric distance and we select the subspace which corresponds to the minimum distance. Note that this also provides an encoding scheme with

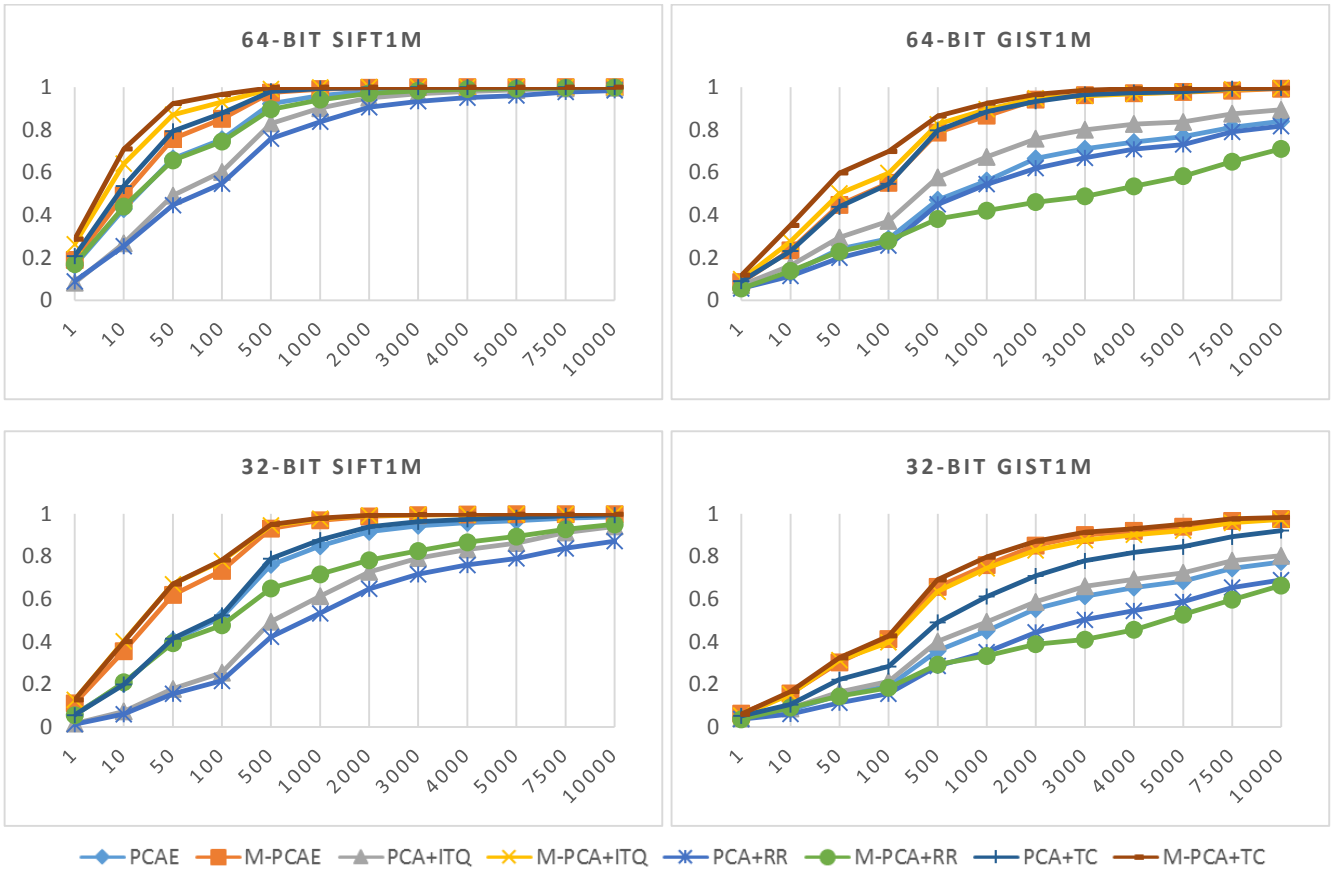

Figure 2: Experimental results. Vertical axis indicates recall@R, while horizontal axis is the R value.

minimum quantization error. The embedding function is formulated in Eq. (11).

$$h_m(\boldsymbol{x}_i) = \underset{c \in C}{\operatorname{argmin}}(d(\boldsymbol{x}_i, c)) \tag{11}$$

where C is the set of all codes from all subspaces and all dimensions. $d(\boldsymbol{x}_i, c)$ is calculated as given in Eq. (8). The flow of the whole binary encoding system is provided in TABLE 1.

TABLE 1
Algorithm of M-PCA Binary Embeddings

Given: X: a set of samples, M: number of PCA's
N_{it}: number of iterations

$\boldsymbol{I}$: PCA indices for all samples, $0 < \boldsymbol{I}_i \leq M$

- Initialize $\boldsymbol{I}$ with M clusters using K-Means
- For N_{it} iterations, $(it = 1{:}\,N_{it})$
  - For each cluster m in M
    - Perform PCA
    - Perform dimension reduction
    - Perform the PCA based binary embedding method.
  - For each sample $\boldsymbol{x}_i$ in $\boldsymbol{X}$
    - Find the nearest subspace.
    - *Update* cluster index $\boldsymbol{I}_i$ in $\boldsymbol{I}$

V. Experiments

We test our proposed method on two publicly available datasets SIFT1M and GIST1M [17]. SIFT1M consists of 1 Million 128-D SIFT vectors and GIST1M includes 1 Million 960-D GIST descriptors. There are 10,000 additional query vectors in SIFT1M and 1,000 queries in GIST1M. We perform exhaustive search on both datasets for all corresponding queries. We use **recall@*R*** measure as our performance metric, which is the recall value for the first R samples in retrieval and we assume the nearest neighbor is the ground truth for each query.

We compare our proposed method with traditional binary embedding methods which use PCA for dimension reduction. The selected baseline methods are PCAE [10], PCA-RR [11], ITQ [11] and TC [12]. We test our system with 32-bit and 64-bit code lengths. The results are presented in Figure 2.

In our experiments we choose the subspace number M as 256. Note that in order to index 256 different subspace, we use 8-bits from our bit allocation budget. For example, for 64-bit coding, we use the first 8-bits for indexing and remaining 56-bits for binary embedding. As it can be seen in the given results, representing the data with M-PCA drastically increases the retrieval performance for with asymmetric distances. In all of the comparisons, M-PCA method performs better than methods with one single PCA.

VI. Conclusion

In this paper, we propose introducing multiple PCA transformations to represent the data for binary embedding methods and prove that it provides a major increase in the performance when asymmetric distances are used. In our future work, we will focus on testing our method on labeled datasets and adapting the same idea for supervised approaches which rely on single PCA transformations.

References

[1] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. J. Comput. Vis.*, vol. 42, no. 3, pp. 145–175, 2001.

[2] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-Sensitive Hashing Scheme Based on P-stable Distributions," in *SCG*, 2004, p. 253.

[3] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognit. Lett.*, vol. 31, no. 11, pp. 1348–1358, Aug. 2010.

[4] K. Terasawa and Y. Tanaka, "Spherical LSH for Approximate Nearest Neighbor Search on Unit Hypersphere," in *WADS*, 2007, pp. 27–38.

[5] J. Buhler, "Efficient Large-Scale Sequence Comparison by Locality-Sensitive Hashing," *Bioinformatics*, vol. 17, pp. 419–428, 2001.

[6] Y. Weiss, A. Torralba, and R. Fergus, "Spectral Hashing," in *NIPS*, 2009, pp. 1753–1760.

[7] P. Li, M. Wang, and J. Cheng, "Spectral hashing with semantically consistent graph for image indexing," *IEEE Trans. Multimed.*, vol. 15, no. 1, pp. 141–152, 2013.

[8] Y. Wang, S. Tang, Y. Zhang, J. Li, and D. Chen, "Fitted spectral hashing," in *Proceedings of the 21st ACM international conference on Multimedia - MM '13*, 2013, pp. 645–648.

[9] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for large-scale search.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 12, pp. 2393–406, Dec. 2012.

[10] A. Gordo, F. Perronnin, Y. Gong, and S. Lazebnik, "Asymmetric distances for binary embeddings.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 1, pp. 33–47, Jan. 2014.

[11] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *CVPR*, 2011, pp. 817–824.

[12] J. Brandt, "Transform coding for fast approximate nearest neighbor search in high dimensions," in *CVPR*, 2010, pp. 1815–1822.

[13] N. Kambhatla and T. K. Leen, "Dimension Reduction by Local Principal Component Analysis," *Neural Comput.*, vol. 9, no. 7, pp. 1493–1516, Oct. 1997.

[14] P. Agarwal and N. Mustafa, "k-Means Projective Clustering," in *SIGMOD*, 2004, pp. 155–165.

[15] V. Gassenbauer, J. Křivánek, K. Bouatouch, C. Bouville, and M. Ribardière, "Improving Performance and Accuracy of Local PCA," *Comput. Graph. Forum*, vol. 30, no. 7, pp. 1903–1910, Sep. 2011.

[16] C. M. Bishop, "Bayesian PCA," in *NIPS*, 1999, vol. 11, pp. 382–388.

[17] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–28, Jan. 2011.

II

K-SUBSPACES QUANTIZATION FOR APPROXIMATE NEAREST NEIGHBOR SEARCH

by

E.C.Ozan, S. Kiranyaz & M. Gabbouj, July 2016

IEEE Transactions on Knowledge and Data Engineering (TKDE), vol. 28, no. 7, pp. 1722-1733.

K-Subspaces Quantization for Approximate Nearest Neighbor Search

Ezgi Can Ozan, Serkan Kiranyaz, *Senior*, *IEEE* and Moncef Gabbouj, *Fellow*, *IEEE*

Abstract— Approximate Nearest Neighbor (ANN) search has become a popular approach for performing fast and efficient retrieval on very large-scale datasets in recent years, as the size and dimension of data grow continuously. In this paper, we propose a novel vector quantization method for ANN search which enables faster and more accurate retrieval on publicly available datasets. We define vector quantization as a multiple affine subspace learning problem and explore the quantization centroids on multiple affine subspaces. We propose an iterative approach to minimize the quantization error in order to create a novel quantization scheme, which outperforms the state-of-the-art algorithms. The computational cost of our method is also comparable to that of the competing methods.

Index Terms— Approximate Nearest Neighbor Search, Binary Codes, Large-Scale Retrieval, Subspace Clustering, Vector Quantization

1 INTRODUCTION

THE Nearest Neighbor (NN) search aims to find a sample in a given dataset that is closest to a given query, which is called the nearest neighbor. It is widely used in different areas of signal processing such as information retrieval, computer vision, machine learning, pattern recognition and recommendation systems. However, the traditional NN search is not tractable for today's very large-scale datasets. Both the search on the dataset and the distance calculation between sample pairs are computationally costly, considering the number of samples and the dimension of the feature space. In order to overcome these limitations of NN search and make it feasible for large-scale problems, Approximate Nearest Neighbor (ANN) search has been proposed [1]. ANN search uses compact representations in order to approximate pair-wise distances between pairs of data points. It has proved to be a viable alternative and has so far achieved promising results [2].

Many of the existing algorithms in this field rely on the concept of "hashing". Hashing methods aim to create binary strings from sample vectors and compare those strings using the Hamming distance representing the proximity neighborhood or some given similarity [3]–[7]. The binary string comparison has also evolved from the simple Hamming distance to asymmetrical distance measures [8], [9] and the usage of look-up tables has enabled more accurate approximations, directing the research on ANN search towards vector quantization [2]. The focus of this paper is on vector quantization based approaches, and the reader is referred to [2] for a more detailed review on hashing.

The idea of quantization goes back to 1980's. Lloyd defined the concept of "good quantization" [10], which is closely related to the K-Means algorithm [11]. Yet Lloyd's quantization method, or K-Means is not directly applicable to large-scale data, for very large number of centroids. For example, considering a quantization using a binary string of 64-bits, the desired number of centroids is 2^{64}. Obviously, it is neither possible to find nor to store such amount of data.

A great improvement on Lloyd's approach for quantization has been proposed by Jegou *et al.* [12] for ANN. In their method called Product Quantization (PQ), the authors divide the sample vector into subvectors and quantize each of them independently using subquantizers. This makes the quantization codebook a Cartesian product, where each centroid in this codebook is represented as a concatenation of the corresponding centroids from the subcodebooks. Therefore, for a small number of subquantizers, while each of them having a feasible number of centroids, obtaining the desired total number of centroids is made possible. Referring to the example above, thanks to the Cartesian product, selecting the number of subquantizers as 8 and the number of centroids for each subquantizer as 256 would be enough to reach 2^{64}. This approach however suffers from the statistical dependency of subvectors, since they are quantized independently.

Another approach for efficient coding on high dimensional vectors has been proposed by Jegou *et al.* [6] and later by Gordo *et al.* [8]. In both papers, the data is decorrelated by applying the Principal Component Analysis (PCA), and its dimension is reduced to a desired value. Then, quantization is applied independently on each of the remaining principal components. This, with a Gaussian distribution assumption, solves the statistical dependency problem among the dimensions of vectors; however, it brings the problem of an unbalanced distribution of information (variance) among dimensions.

As a result of PCA, the principal components are ranked in a decreasing order according to the corresponding variances,

- *Ezgi Can Ozan, is with the Department of Signal Processing, Tampere University of Technology, Tampere, Finland (e-mail: ezgi.ozan@tut.fi).*
- *Prof. Serkan Kiranyaz, is with Electrical Engineering Department, College of Engineering, Qatar University, Qatar. (e-mail: mkiranyaz@qu.edu.qa).*
- *Prof. Moncef Gabbouj, is with the Department of Signal Processing, Tampere University of Technology, Tampere, Finland (e-mail: moncef.gabbouj@tut.fi).*

yet each component is quantized by two centroids only. Gong *et al.* [13] propose a two-step method called Iterative Quantization (ITQ). In the first step, a PCA transformation is applied for dimension reduction as in [6], while in the second step, an orthogonal rotation on the data is applied iteratively for a balanced distribution of the variance among the principal components. The data are quantized on the principal components of the rotated space independently. While orthogonal rotations preserve the Euclidean distance between pairs of samples, here again, the problem of statistical dependency reappears, since the dimensions are no longer decorrelated.

Brandt in [14] proposes a method called Transform Coding (TC), to balance the variance corresponding to each code separately after PCA. TC is a special case of PQ, where each dimension itself is a subvector. However, in TC, each dimension is allocated a variable number of bits, and a scalar quantization is performed on each principal component independently. Following the rotation idea in [13], Optimized Product Quantization [15] (OPQ) and Cartesian K-Means (CKM) [16] both produce an improvement over PQ by applying an iterative optimization process in order to balance the dimension variances. In [17] Heo *et al.* improve OPQ by encoding the distances to centroids separately in their algorithm called Distance Encoded Product Quantization (DEPQ). Locally Optimized Quantization (LOPQ) [18] introduces local optimization before OPQ and further improves the performance.

Recently summation based multi-stage vector quantization methods such as Optimized Cartesian K-Means (OCK) [19], Additive Quantization (AQ) [20], Composite Quantization (CQ) [21] and (Optimized) Tree Quantization (OTQ) [22] which aim to use the summation of several dictionary items to represent the approximation of a vector, have been proposed. These methods produce the state-of-the-art results although the theory behind such quantization methods has been well studied in the past [23].

Many of the proposed methods so far transform or project the data into a new (sub)space, where vector dimensions are reduced, reordered or rotated using PCA. Decorrelating the data using a single PCA step may not bring the desired statistical independency among dimensions, especially if the data do not follow a Gaussian distribution, which is the core inherent assumption of PCA. A better transformation however, may be designed by representing the data with more than one subspace. Local-PCA [24], [25], K-Means Projective Clustering [26] and Bayesian PCA [27] all propose different solutions to this problem based on PCA. In our recent study entitled M-PCA Binary Embedding (MPCA-E) [28], we have also shown that using traditional PCA based embedding approaches, such as [6], [8], [13], [14], with multiple PCAs instead of only a single PCA, the performance is improved significantly. In this paper this result is taken one step further, by developing an iterative approach to obtain the affine subspaces and codebooks at the same time. In this way, the proposed method achieves lower quantization error, which leads to a better encoding scheme with state-of-the-art performance. The main contributions of the proposed method are the following:

- Vector quantization is defined as a multiple subspace learning problem, where the objective is to minimize the quantization error of the training samples in the learnt subspaces, while also minimizing the projection error of the samples to the corresponding subspaces.
- An optimization problem that jointly minimizes both errors defined above is formulated as an iterative process.
- A simple, yet effective scheme for faster sample encoding is proposed, by efficiently selecting a limited number of subspaces, thus decreasing the computational cost required for evaluating all possible encodings.
- The proposed approach is evaluated on publicly available datasets, and shown to achieve state-of-the-art performance.

The rest of the paper is organized as follows: The problem formulation is given in **Section 2** and the detailed steps in the proposed method are described in **Section 3**. In **Section 4**, experiments on publicly available benchmark datasets are presented and in **Section 5**, the parameter selection, scalability and the computational and storage costs of the method are discussed. Finally in **Section 6**, the paper is concluded.

2 Problem Formulation

The problem of vector quantization for ANN search can be formulated as follows: Given a set of N D-dimensional vectors $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$ and a query vector $\boldsymbol{q}$, ($\boldsymbol{q}, \boldsymbol{x}_i \in \mathbb{R}^D$) the nearest neighbor search aims to find $\underset{i}{\operatorname{argmin}}\, d(\boldsymbol{x}_i, \boldsymbol{q})$, where $d(\boldsymbol{x}_i, \boldsymbol{q})$ is a distance computed between $\boldsymbol{x}_i$ and $\boldsymbol{q}$. The approximate nearest neighbor search however aims to find a point $\boldsymbol{x}_{ANN}$ such that,

$$d(\boldsymbol{x}_{ANN}, \boldsymbol{q}) \leq (1+\epsilon)\, d(\boldsymbol{x}_{NN}, \boldsymbol{q}) \tag{1}$$

where $\boldsymbol{x}_{NN}$ is the true nearest neighbor and $\epsilon > 0$. A quantizer Q quantizes the vector to its corresponding codevector $\boldsymbol{c}_j \in \mathbb{R}^D$ within the codebook $\boldsymbol{C} = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_M\}$, where M is the number of codevectors. The mean squared quantization error MSE_Q is defined by

$$MSE_Q = \frac{1}{N}\sum_{i}^{N} \|\boldsymbol{x}_i - Q(\boldsymbol{x}_i)\|^2 \tag{2}$$

The equation in (2) can be extended to (3), where $\boldsymbol{R} \in \mathbb{R}^{D\times L}$ ($L \leq D$) is an orthogonal projection matrix and $\boldsymbol{R}^{\perp}$ spans the orthogonal complement of the range space of $\boldsymbol{R}$. With this extension, it is possible to define MSE_Q also for [6], [8], [13], [14], where the quantization is performed after dimension reduction or transformation. Equation (2) is a special case of (3), where $\boldsymbol{R} = \boldsymbol{I}$ and $D = L$.

$$MSE_Q = \frac{1}{N}\sum_{i}^{N} \left(\|\boldsymbol{R}^T\boldsymbol{x}_i - Q(\boldsymbol{R}^T\boldsymbol{x}_i)\|^2 + \left\|\boldsymbol{R}^{\perp T}\boldsymbol{x}_i\right\|^2 \right) \tag{3}$$

It is clear from (3) that the quantization error MSE_Q is directly affected by the selection of the projection matrix $\boldsymbol{R}$. The second term in the summation adds a non-negative value to the quantization error, unless $\boldsymbol{R}^{\perp}$ is zero. No matter how close codevector $\boldsymbol{c}_j$ is to the sample $\boldsymbol{x}_i$, there is a non-zero quantization error depending on $\boldsymbol{R}^{\perp}$ and $\boldsymbol{x}_i$. Thus, in order to minimize this error, an appropriate projection matrix $\boldsymbol{R}$ should be chosen.

Gong *et al.* in [13] propose a method which performs an orthogonal rotation in the feature space in order to minimize the quantization error. First, they apply PCA on the data and iteratively rotate the principal components in order to match the samples with their corresponding codevectors, resulting in smaller quantization error. Let $\boldsymbol{R} \in \mathbb{R}^{D\times L}$ be the PCA dimension reduction matrix. The error given in (4) is minimized by introducing another orthogonal rotation matrix $\boldsymbol{P} \in \mathbb{R}^{L\times L}$ and following the Procrustean approach [13].

$$MSE_Q = \frac{1}{N}\sum_{i}^{N}\Big(\|(\boldsymbol{RP})^T\boldsymbol{x}_i - Q((\boldsymbol{RP})^T\boldsymbol{x}_i)\|^2 + \left\|\boldsymbol{R}^{\perp T}\boldsymbol{x}_i\right\|^2\Big) \tag{4}$$

In [13], Gong *et al.* proposed a method to select $\boldsymbol{R}$ in (3), but the selection aims to improve mainly the first term of the summation. The second term of the summation is also minimized with the selection of $\boldsymbol{R}$ as PCA dimension reduction matrix, since the principal components are ordered with decreasing variance. In [15], [16] and [18]; the authors are inspired from the rotation of [13] to fit the data better to codevectors, and combine this approach with PQ. They follow the same iterative approach to find the optimal rotation. In [16], the quantization error is defined as given in (5), where H is the number of subvectors and the subscript h represents the matrix for the subvector in question. $\boldsymbol{D}$ is a diagonal scaling matrix.

$$MSE_Q = \frac{1}{N}\sum_{i}^{N}\sum_{h}^{H}\left\|\boldsymbol{R}_h^T\boldsymbol{x}_i - \boldsymbol{D}_h Q_h\left(\boldsymbol{R}_h^T\boldsymbol{x}_i\right)\right\|^2 + \left\|\boldsymbol{R}^{\perp T}\boldsymbol{x}_i\right\|^2 \tag{5}$$

The algorithm first keeps $\boldsymbol{R}$ fixed and solves for $\boldsymbol{D}$ and Q, then keeping $\boldsymbol{D}$ and Q constant, it optimizes $\boldsymbol{R}$. Since a full rank $\boldsymbol{R} \in \mathbb{R}^{D\times D}$ is used, the second term in the summation is 0. Although putting a rank constraint on $\boldsymbol{R}$ for high dimensional data is suggested in [16], it is not clearly stated how to decrease the transformation error, which will always be non-zero as long as $rank(\boldsymbol{R}) < D$.

Brandt in [14] selects the projection matrix $\boldsymbol{R} \in \mathbb{R}^{D\times L}$ as the PCA dimension reduction matrix, similar to [13]. However, instead of searching for a better $\boldsymbol{R}$, the bits are distributed non-uniformly among dimensions using the fact that the variances are sorted in a decreasing order for each dimension. That is, multiple centroids are assigned to the dimensions with high variances, while some of the dimensions with lower variances are omitted.

As mentioned above, the necessity of searching for an appropriate matrix $\boldsymbol{R}$ is evident, but so far the researchers have limited themselves to a single transformation matrix for the purpose of minimizing the quantization error. However, a single matrix may not be sufficient to transform the data into a new space where a better representation is possible. In our previous study [28], a two-step solution to this problem is proposed. First the whole space is divided into subspaces using multiple PCAs. Then for each subspace, a PCA based subquantizer is trained. This approach provides a better representation of the data and reduces the total transformation error at the same time. In [18], the authors also propose a similar approach, by dividing the data into local clusters and training quantizers on each cluster separately. However, taking those two steps of clustering and quantization into account separately yields a suboptimal solution. For this reason, in this paper, an iterative joint optimization scheme for both steps is proposed.

3 The Proposed Method

As explained in the previous section, the proposed solution starts by extending the equation (3) further introducing multiple affine subspaces into the quantization system. Each subspace is allowed to have a different number of dimensions and a separate *subquantizer* is present in each subspace. Let $\boldsymbol{\mathcal{F}}_k \in \mathbb{R}^{L_k},\ k = \{1, \dots, K\}$ be a subspace defined by the affine shift vector $\boldsymbol{\mu}_k \in \mathbb{R}^D$ and the projection matrix $\boldsymbol{R}_k \in \mathbb{R}^{D\times L_k}$, where L_k is the number of dimensions of the subspace $\boldsymbol{\mathcal{F}}_k$. Let $\boldsymbol{X}_k \subset \boldsymbol{X}$ be a subset of samples (a cluster) such that $\bigcup_{k=1}^{K}\boldsymbol{X}_k = \boldsymbol{X}$ and $\boldsymbol{X}_k \cap \boldsymbol{X}_j = \emptyset$ where $k \neq j$. $\boldsymbol{\mathcal{F}}_k$ is the selected affine subspace for each sample $\boldsymbol{x}_i \in \boldsymbol{X}_k$. Finally, let Q_k be a subquantizer which maps a given vector $\boldsymbol{x}_i$ to its corresponding codevector $\boldsymbol{c}_{j,k} \in \mathbb{R}^{L_k}$ within the codebook $\boldsymbol{C}_k = \{\boldsymbol{c}_{1,k}, \dots, \boldsymbol{c}_{M_k,k}\}$, where M_k is the number of codevectors for the codebook $\boldsymbol{C}_k$. We redefine the mean squared quantization error MSE_Q, as formulated in (6). Note that (3) is a special case of (6), where $K = 1$ and $\boldsymbol{\mu}_k = \boldsymbol{0}$.

$$MSE_Q = \frac{1}{N}\sum_{k}^{K}\sum_{\boldsymbol{x}_i\in\boldsymbol{X}_k}\Big(\left\|\boldsymbol{R}_k{}^T(\boldsymbol{x}_i - \boldsymbol{\mu}_k) - Q_k\left(\boldsymbol{R}_k{}^T(\boldsymbol{x}_i - \boldsymbol{\mu}_k)\right)\right\|^2 + \left\|\boldsymbol{R}_k{}^{\perp T}(\boldsymbol{x}_i - \boldsymbol{\mu}_k)\right\|^2\Big) \tag{6}$$

The minimization of (6) for a fixed $K > 1$ is an NP-Hard problem to solve. So in the proposed method, an approximate solution to minimize the error term given in (6) is proposed by following an iterative approach. Each iteration consists of four steps:

i. *determination of subspaces,*
ii. *bit allocation,*
iii. *scalar quantization,*
iv. *cluster updates.*

With this iterative approach, first it is aimed to find K suitable affine subspaces $\boldsymbol{\mathcal{F}}_k$, each defined by a matrix $\boldsymbol{R}_k$ and an affine shift vector $\boldsymbol{\mu}_k$. Then for each $\boldsymbol{\mathcal{F}}_k$, the bits are distributed among dimensions and the subquantizer Q_k is obtained. Finally the cluster index for each sample is updated by the most suitable cluster. The input samples are initially assigned to K different clusters. The initial cluster indices are obtained

by using K-Means. After the initialization, the iterations begin.

3.1 Determination of Subspaces

In the proposed method, an affine subspace $\mathcal{F}_k$ is defined for each cluster $\boldsymbol{X}_k$. First, the mean of $\boldsymbol{X}_k$ is calculated to obtain the affine shift vector $\boldsymbol{\mu}_k = \frac{1}{N_k}\sum_{\boldsymbol{x}_i \in \boldsymbol{X}_k} \boldsymbol{x}_i$; and then PCA is applied using the samples $\boldsymbol{X}_k$, in order to obtain the transformation matrix $\boldsymbol{R}_k$, as proposed in [24]–[26]. Here it should be emphasized that each subspace $\mathcal{F}_k$ may have a variable number of dimensions L_k and this number is constrained only by the number of bits and their allocation among dimensions. Therefore, the number of dimensions of a subspace is determined according to the bit allocation strategy, which is explained in **Section 3.3**. In the next section the selection of the scalar quantization for the proposed method is defined and justified.

3.2 Scalar Quantization

In the formulation of the problem above, it has been stated that multiple affine subspaces are introduced in order to minimize the error generated by the projection onto a subspace. Since multiple PCAs are used to generate the projection matrices, the statistical dependencies between dimensions have been minimized [24]–[26], i.e., quantization can be performed independently on each dimension and the codevector can be obtained as a Cartesian product of quantized values similar to [8],[13] or [14].

Let $\hat{\boldsymbol{x}}_{i,k}$ be the projection of $\boldsymbol{x_i}$ onto the corresponding subspace k, i.e., $\hat{\boldsymbol{x}}_{i,k} = \boldsymbol{R}_k{}^T(\boldsymbol{x}_i - \boldsymbol{\mu}_k)$. For the l^{th} dimension $\hat{\boldsymbol{x}}_{i,k}^l \in \mathbb{R}^1$ of a projected vector $\hat{\boldsymbol{x}}_{i,k}$, and a centroid $\dot{\boldsymbol{c}}_k^l \in \mathbb{R}^1$, i.e., $\dot{\boldsymbol{c}}_k^l = Q_k\left(\hat{\boldsymbol{x}}_{k,i}\right)^l$; the mean squared scalar quantization error on dimension l, $MSE_{Q_s}^l$, can be formulated as:

$$MSE_{Q_s}^l = \frac{1}{N_k}\sum_{\boldsymbol{x}_i \in \boldsymbol{X}_k} \left\|\hat{\boldsymbol{x}}_{i,k}^l - \dot{\boldsymbol{c}}_k^l\right\|^2 \qquad (7)$$

Note that, if $\dot{\boldsymbol{c}}_k^l$ is selected as the expected value of $\boldsymbol{X}_k^l$, i.e., $\dot{\boldsymbol{c}}_k^l = \frac{1}{N_k}\sum_{\boldsymbol{x}_i \in \boldsymbol{X}_k} \hat{\boldsymbol{x}}_{i,k}^l$, then $MSE_{Q_s}^l$ corresponds to the variance of $\boldsymbol{X}_k^l$. Since the variance among the dimensions is not balanced after PCA, the bits are distributed among dimensions non-uniformly, as also done in [14]. After the number of bits for each dimension is determined as described in the next section, Max-Lloyd quantization is applied independently for each dimension, obtaining 2^{b_l} centroids, where b_l is the number of bits assigned for dimension l. The centroids are stored to create a quantization codebook.

3.3 Bit Allocation

As mentioned earlier, since the variance is not distributed uniformly among the dimensions after PCA (the dimensions are ordered with decreasing variances), a non-uniform bit allocation scheme is proposed. In this paper a bit-wise adaptation of the *Modified-d´Hondt* method [29] is proposed, which is a widely used seat distribution method in parliamentary elections. If the bit allocation was a parliamentary election, then bits would be the seats in the parliament and the standard deviations would be the votes for each party. The standard deviations of discarded dimensions would correspond to votes given to parties who could not enter the parliament.

In *d'Hondt* method, the votes for the candidates are stored in a vector. These values are divided by the number of seats already assigned to the candidates. After the division, the candidate with the highest value is given another seat. In the proposed adaptation, candidates are the dimensions, votes are the standard deviations and number of seats correspond to the number of centroids. The algorithm starts by taking the square root of the eigenvalues to obtain the standard deviations. For each dimension, the standard deviation is divided by the number of centroids corresponding to that dimension. For the dimension that yields the largest division, a bit is appointed and this continues until all bits are assigned. Note that, for $\boldsymbol{b}_l$ bits appointed to dimension l, the number of centroids is $2^{\boldsymbol{b}_l}$.

The *d'Hondt* method penalizes the dimensions for each bit they receive, but in order to make it harder for dimensions with low standard deviations to get their first bit, hence keeping the dimension of the transformed vector as small as possible, *d'Hondt* method is *modified* and the standard deviations are divided by $\sqrt{2}$ instead of 1 when $\boldsymbol{b}_l$ is equal to 0. The pseudo-code for the bit allocation is given in **TABLE 1**, and an example demonstrating the bit distribution steps for 4 bits on a 4-dimensional vector is given in **TABLE 2**. Also the difference between the *d'Hondt* and the *Modified d'Hondt* methods, and their comparison with TC [14] are presented in **Fig. 1** for 32-bits.

TABLE 1
PSEUDO-CODE FOR MODIFIED-D'HONDT METHOD

Given: $\boldsymbol{V}$: vector of standard deviations, B: number of bits
Return: $\boldsymbol{b}$: number of allocated bits for each dimension

$\boldsymbol{b}_l$: The number of bits assigned to dimension l
$\boldsymbol{V}_l$: The standard deviation for dimension l
$\boldsymbol{p}_l$: The standard deviation per centroid for dimension l

- $\boldsymbol{b} = \boldsymbol{0}$
- *while* $\sum \boldsymbol{b}_l < B$
  - *for each* dimension l
    - $if\,(\boldsymbol{b}_l = 0)$
      - $\boldsymbol{p}_l = \boldsymbol{V}_l/\sqrt{2}$
    - *else*
      - $\boldsymbol{p}_l = \boldsymbol{V}_l/\sqrt{2^{\boldsymbol{b}_l}}$
  - Allocate 1 bit to $\max_l \boldsymbol{p}_l$, $(\boldsymbol{b}_l = \boldsymbol{b}_l + 1)$

TABLE 2
BIT ALLOCATION EXAMPLE FOR 4 BITS ON A 4D VECTOR.

		Principal Comp. 1	Principal Comp. 2	Principal Comp. 3	Principal Comp. 4
	Std. Dev.	100	75	50	10
	Allocate:	*0*	*0*	*0*	*0*
Iter. 1	Divide:	$\frac{100}{\sqrt{2}} = 70.7$	$\frac{75}{\sqrt{2}} = 53.1$	$\frac{50}{\sqrt{2}} = 35.4$	$\frac{10}{\sqrt{2}} = 7.07$
	Allocate:	*1*	*0*	*0*	*0*
Iter. 2	Divide:	$\frac{100}{2^1} = 50$	$\frac{75}{\sqrt{2}} = 53.1$	$\frac{50}{\sqrt{2}} = 35.4$	$\frac{10}{\sqrt{2}} = 7.07$
	Allocate:	*1*	*1*	*0*	*0*
Iter. 3	Divide:	$\frac{100}{2^1} = 50$	$\frac{75}{2^1} = 37.5$	$\frac{50}{\sqrt{2}} = 35.4$	$\frac{10}{\sqrt{2}} = 7.07$
	Allocate:	*2*	*1*	*0*	*0*
Iter. 4	Divide:	$\frac{100}{2^2} = 25$	$\frac{75}{2^1} = 37.5$	$\frac{50}{\sqrt{2}} = 35.4$	$\frac{10}{\sqrt{2}} = 7.07$
	Allocate:	*2*	*2*	*0*	*0*

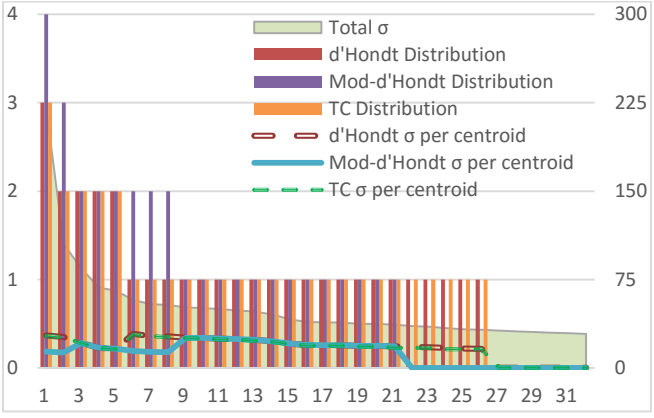

Fig. 1: Bit allocation as given in TC [14], by d'Hondt and Modified d'Hondt methods. (left vertical axis: bits per dimension, right vertical axis: standard deviation, horizontal axis: dimensions)

As it can be seen in **Fig. 1**, the *Modified d'Hondt* method emphasizes more the dimensions with higher variances. Since there is a limited number of bits, assigning multiple bits to one dimension means that another dimension is discarded from quantization, so this dimension can also be removed from the subspace. In other words, the reduced number of dimensions L_k for a subspace is the number of dimensions which has at least one allocated bit.

3.4 Cluster Updates

Once the subspaces and their number of dimensions are established, the quantizers are obtained and each sample from the training set is assigned to its new cluster. The new subspace of a sample $\boldsymbol{x}_i$ is determined by (8) and each sample is assigned to the cluster that gives the lowest quantization error.

$$\underset{k}{\operatorname{argmin}} \left(\left\| \boldsymbol{R}_k{}^T (\boldsymbol{x}_i - \boldsymbol{\mu}_k) - Q_k\left(\boldsymbol{R}_k{}^T (\boldsymbol{x}_i - \boldsymbol{\mu}_k)\right) \right\|^2 + \left\| (\boldsymbol{x}_i - \boldsymbol{\mu}_k) - \boldsymbol{R}_k \boldsymbol{R}_k{}^T (\boldsymbol{x}_i - \boldsymbol{\mu}_k) \right\|^2 \right) \quad (8)$$

Note that the second term in (8) is identical to the second term in (6). This term represents the distance of a sample $\boldsymbol{x}_i$ to the affine subspace $\boldsymbol{\mathcal{F}}_k$, which is defined by matrix $\boldsymbol{R}_k$ and the affine shift vector $\boldsymbol{\mu}_k$, as follows:

$$d(\boldsymbol{x}_i, \boldsymbol{\mathcal{F}}_k) = \left\| \boldsymbol{R}_k{}^{\perp T} (\boldsymbol{x}_i - \boldsymbol{\mu}_k) \right\| \quad (9)$$

This distance can be equivalently calculated using (10) if $\boldsymbol{R}_k$ is an orthogonal projection matrix. Here it is recommended to use (10) instead of (9), because (9) requires the storage of $\boldsymbol{R}_k{}^{\perp}$, which brings additional memory cost, especially when $D \gg L_k$.

$$d(\boldsymbol{x}_i, \boldsymbol{\mathcal{F}}_k) = \left\| (\boldsymbol{x}_i - \boldsymbol{\mu}_k) - \boldsymbol{R}_k \boldsymbol{R}_k{}^T (\boldsymbol{x}_i - \boldsymbol{\mu}_k) \right\| \quad (10)$$

3.5 Filtering Outliers

In order to prevent early convergence to a local minimum, in the proposed method, a percentage of the samples are filtered out according to the quantization error, before updating the corresponding PCA, similar to [26]. It starts with 25% and is reduced by 1% at each iteration.

Up to now, in Section 3, the proposed iterative approach to obtain the codevectors while minimizing the error in (6) is described. The whole training algorithm is given as a pseudo-code in **TABLE 3**.

TABLE 3
TRAINING FOR K-SUBSPACE QUANTIZATION

Given: $\boldsymbol{X}$: set of samples, K: number of subspaces
N_{it}**:** number of iterations
Return: $\boldsymbol{C}_k$, $\boldsymbol{R}_k$ and $\boldsymbol{\mu}_k$ for all k : codebooks, transformation matrices and affine shift vectors for all subspaces

$\boldsymbol{cl}$: Vector of cluster indices for all samples, $0 < \boldsymbol{cl}_i \leq K$

- Initialize $\boldsymbol{cl}$ for K clusters using K-Means
- For N_{it} iterations,
  - *for each* cluster $\boldsymbol{X}_k$
    - Perform PCA to obtain $\boldsymbol{R}_k$ and $\boldsymbol{\mu}_k$
    - Distribute bits as in Section 3.3
    - Perform dimension reduction
    - Obtain the codebook $\boldsymbol{C}_k$
  - *for each* sample $\boldsymbol{x}_i$ in $\boldsymbol{X}$
    - Find the cluster with the minimum quantization error.
    - Update cluster index $\boldsymbol{cl}_i$
  - Filter outliers.

3.6 Sample Encoding

The process of sample encoding is performed as follows: The given sample $\boldsymbol{v} \in \mathbb{R}^D$ is projected onto the subspace $\boldsymbol{\mathcal{F}}_k$ to obtain $\hat{\boldsymbol{v}}_k \in \mathbb{R}^{L_k}$ which corresponds to the minimum quantization error, as given in (8). For each dimension l, the nearest centroid $\boldsymbol{c}_{k,j}^l$ among 2^{b_l} centroids is determined by the subquantizer. The binary string representing the chosen centroids is concatenated to the index k of the subspace $\boldsymbol{\mathcal{F}}_k$ in order to generate the final binary code.

3.7 Speeding up the Encoding Process

In order to calculate the quantization error for all subspaces, the sample should be projected onto each subspace and quantized by the corresponding subquantizer. However, to improve the encoding speed, instead of looking for the nearest code in all K subspaces, the distance between the given vector $\boldsymbol{v}$ and the affine shift vector $\boldsymbol{\mu}_k$ of each affine subspace is calculated and the first $\mathcal{K}$ subspaces with the smallest distances are selected. Then $\boldsymbol{v}$ is only projected onto those subspaces and the quantization errors are calculated. Experimentally it has been observed that, for a limited number of subspace projections, almost the same quantization error (less than 1% difference) can be obtained. Using this approach, the encoding process is accelarated by approximately 16 times. The pseudo-code for encoding a new sample vector is presented in **TABLE 4**.

TABLE 4
ENCODING A NEW SAMPLE

Given: $\boldsymbol{v}$: a sample to be quantized,
Return: $\boldsymbol{\varsigma}$: binary code string

- Select $\mathcal{K}$ subspaces with closest centers to $\boldsymbol{v}$
- $for\ each\ k$ in $\mathcal{K}$
  - Project sample v to the subspace $\boldsymbol{\mathcal{F}}_k$ and calculate the quantization error as in (8).
- Append the binary string which corresponds to the smallest distance, to the binary string $\boldsymbol{\varsigma}$
- Append index k to the binary string $\boldsymbol{\varsigma}$

3.8 Distance Approximation for ANN

In this study, an asymmetric distance calculation method is proposed in order to approximate the distance between a query vector and a code. Asymmetric distances have proved to outperform symmetric approaches [8], but it is required to have the uncompressed query vector at hand. The proposed distance calculation method is formulated in (11), where $\hat{\boldsymbol{x}}_k$ is the projection of $\boldsymbol{x}$ onto the corresponding subspace, i.e., $\hat{\boldsymbol{x}}_k = \boldsymbol{R}_k{}^T(\boldsymbol{x} - \boldsymbol{\mu}_k)$. $d(\boldsymbol{x}, \hat{\boldsymbol{x}}_k)$ represents the distance to a subspace, as given in (6) and $d(\hat{\boldsymbol{x}}_k, \boldsymbol{c}_k)$ corresponds to the Euclidean distance to a code $\boldsymbol{c}_k$ in $\boldsymbol{\mathcal{F}}_k$. $d(\hat{\boldsymbol{x}}_k, \boldsymbol{c}_k)$ is formulated as in (12).

$$d(\boldsymbol{x}, \boldsymbol{c}_k) = \sqrt{d(\boldsymbol{x}, \hat{\boldsymbol{x}}_k)^2 + d(\hat{\boldsymbol{x}}_k, \boldsymbol{c}_k)^2} \tag{11}$$

$$d(\hat{\boldsymbol{x}}_k, \boldsymbol{c}_k) = \left\| \boldsymbol{R}_k{}^T(\boldsymbol{x} - \boldsymbol{\mu}_k) - \boldsymbol{c}_k \right\| \tag{12}$$

3.9 Relations with Other Methods

In this section the links between the proposed method and other related methods from the literature are discussed in order to emphasize the novelties of the proposed method.

3.9.1 Transform Coding

As mentioned in **Sections 1** and **2**, Transform Coding (TC) [14] is a vector quantization method which proposes to perform quantization as a Cartesian product of subquantizers trained independently for each principal component. TC is a special case of the proposed algorithm, where the number of subspace clusters $K = 1$. Both algorithms perform scalar quantization on principal components and allocate bits among dimensions non-uniformly. The novelty of the proposed method comes from the definition and minimization of the error term in (6). The proposed method introduces multiple subspaces and iteratively searches for the quantization centers in those subspaces while jointly minimizing the quantization error. Since multiple subspace representation provides a minimized distance to a subspace for each sample in the dataset, this distance can be introduced in the distance approximation for ANN, as explained in **Section 3.8**. However, in TC, this distance does not have any effect since it is constant for all samples and does not change ANN search order. In the proposed approach an improved bit allocation procedure different than TC is presented, as explained in **Section 3.3**. The performance comparison of both methods is also given in **Section 4**.

3.9.2 Product Quantization

Due to the links of the proposed method with TC, a relation with Product Quantization (PQ) [12] can also be mentioned. Both methods define subspaces but the two definitions are quite different. In PQ, subspaces are defined using subvectors, and subquantizers quantize these subvectors and then a Cartesian product is calculated in order to obtain the final vector. However, in the proposed method, subspaces are defined by affine shift vectors and transformation matrices. The proposed method does not involve any Cartesian product calculations as in PQ, and the subquantizers are independent from each other in that sense.

3.9.3 Locally Optimized Product Quantization

If each cluster in Locally Optimized Product Quantization (LOPQ) [18] is considered as an affine (sub)space with D dimensions, a relation can be established between LOPQ and the proposed method. However, the two methods are different in many aspects. Firstly, in the proposed method the dimensions of subspaces are allowed to be less than D, and each subspace may have a different number of dimensions. In other words, the clustering in LOPQ is a specific case, where $L = D$, while the proposed method is more generic, allowing $L \leq D$.

Another difference between the two methods is the purpose of clustering. LOPQ achieves localization by a coarse quantizer first, and later quantizes the residuals separately, using the coarse quantizer for indexing. In the proposed method, however, the subspace clustering is not a coarse quantization step, but part of an iterative approach to obtain different quantization centers located in different affine subspaces. The samples are quantized to the corresponding nearest cluster centers, whereas in LOPQ, they are quantized to the nearest subspace first then to the nearest center within that space.

Finally, in LOPQ, after coarse quantization, each cluster is further quantized by a local Optimized Product Quantizer (OPQ) [15], while in the proposed method, the quantization centers are searched within different subspaces using a variant of Transform Coding (TC) [14]. Both OPQ and TC rely

on PCA, however, since bits are allcoated non-uniformly in TC, it has a natural dimension reduction property, while OPQ uses all dimensions. This property of TC is quite beneficial for the proposed method as the dimension reduction is combined with subspace clustering, i.e., the dimension of the subspace in question is decided according to the proposed bit allocation method. By using such subquantizers, the required storage overhead is reduced significantly. For example, for the GIST feature, for each subquantizer, one needs to store a 960x960 PCA transformation matrix for OPQ, while the TC variant requires only a matrix of 960x40 (64-bit coding, on average).

3.9.4 M-PCA Binary Embedding

As stated above, in our recent study [28] (MPCA-E), we have further optimized the second error term in (3), by using multiple PCA clustering as a preprocessing step. Then we train a subquantizer for each cluster. However, in this paper, we take it one step further by performing the optimization of the quantizers and the subspaces jointly, with the help of the proposed iterative approach.

In MPCA-E, the data is clustered according to the distance of samples to subspaces. This reduces the transformation error, which also affects the total quantization error. However, in this approach, clustering is performed using the quantization error obtained in the corresponding subspaces. In other words, each sample is assigned to the cluster that results in the lowest quantization error. This iterative approach decreases the overall quantization error directly.

Since multiple PCA clustering is a preprocessing step for MPCA-E, all subspaces have the same predefined dimension. Yet, in this paper, thanks to the joint optimization scheme, each subspace is allowed to have different number of dimensions, hence removing an important restriction from the optimization process. Furthermore, in this study, a subspace selection scheme is developed, which does not require projection onto all subspaces. This improves the encoding speed of the proposed method considerably.

Finally, a variant of Transform Coding is developed for a non-uniform bit allocation in this paper, while (MPCA-E) focuses on and compares only traditional PCA based methods. All these provide a much better quantization scheme, with a lower quantization error and a state-of-the-art performance, as will be presented in the experiments.

3.9.5 Performance Evaluation Using a Toy Example

The application of the proposed method along with a set of recent methods from the literature on a toy example is shown in Fig. 2. Each method is tested on a small 2D dataset, and 2-bit quantization is performed. For LOPQ and KSSQ, the number of subquantizers is 2 and for K-Means there are obviously 4 centroids. Samples encoded by different codevectors, are presented with different colors. Since these methods are designed for large-scale datasets with a high number of dimensions, this toy example does not provide a quantization performance comparison but it is beneficial to visualize the main differences between the methods. For example, it is evident that TC uses only one dimension to perform the quantization, since the variance in the first principal component is much greater than the second one. Another observation could be the rotation in OPQ with respect to PQ, as the principal components are not aligned with the coordinate axes. Since the number of subquantizers is 2, LOPQ first coarsely divides the data into two clusters, then each cluster is quantized with an OPQ. The proposed method, KSSQ, also starts with the same two subspaces, but then iterates to minimize the quantization error, and yields a quantization scheme more similar to the one obtained by K-Means.

3.9.6 Relations with Summation Based Algorithms

As mentioned in **Section 1**, recent state-of-the-art methods use summation based quantization as in [19]–[22]. In these methods, a quantized vector is represented as a sum of several centroids. This representation can be formulated as given in (13). Here Q_k is the subquantizer which quantizes a given vector $\boldsymbol{x}$ to one of the centroids in the codebook k.

$$Q(\boldsymbol{x}) = \sum_{k=1}^{K} Q_k(\boldsymbol{x}) \tag{13}$$

Among these methods, Optimized Cartesian K-Means (OCKM) [19] starts with the rotation of the Cartesian K-Means, then instead of chosing one centroid to quantize the given vector in the subspace in question, the quantized subvector is represented by the addition of two centroids. Additive Quantization (AQ) [20] proposes the addition of codevectors obtained from many different codebooks, but since the computational complexity is quadratically dependent on the number of subcodebooks, an Additive Product Quantization (APQ) [20] method is proposed, which performs Additive Quantization on subspaces of Product Quantization seperately. Composite Quantization (CQ) [21] proposes an additional constraint on the generation of centroids in order to reduce the asymmetric distance calculation complexity. Moreover, Tree Quantization (TQ) [22] brings a graph structure to codewords called the “coding tree”, where each vertex corresponds to a subcodebook of AQ. Each dimension of the given vector is assigned to an edge, resulting in disjoint edges. So any codewords coming from any codebooks, which are not adjacent in the tree, are orthogonal. Finally a global rotation as in [15] for further optimization is proposed resulting in a variant of TQ called Optimized Tree Quantization (OTQ).

The main difference between such methods and the proposed method is, while summation based methods require the addition of the codevectors from subcodebooks, in KSSQ, the most suitable one among them is selected, i.e.,

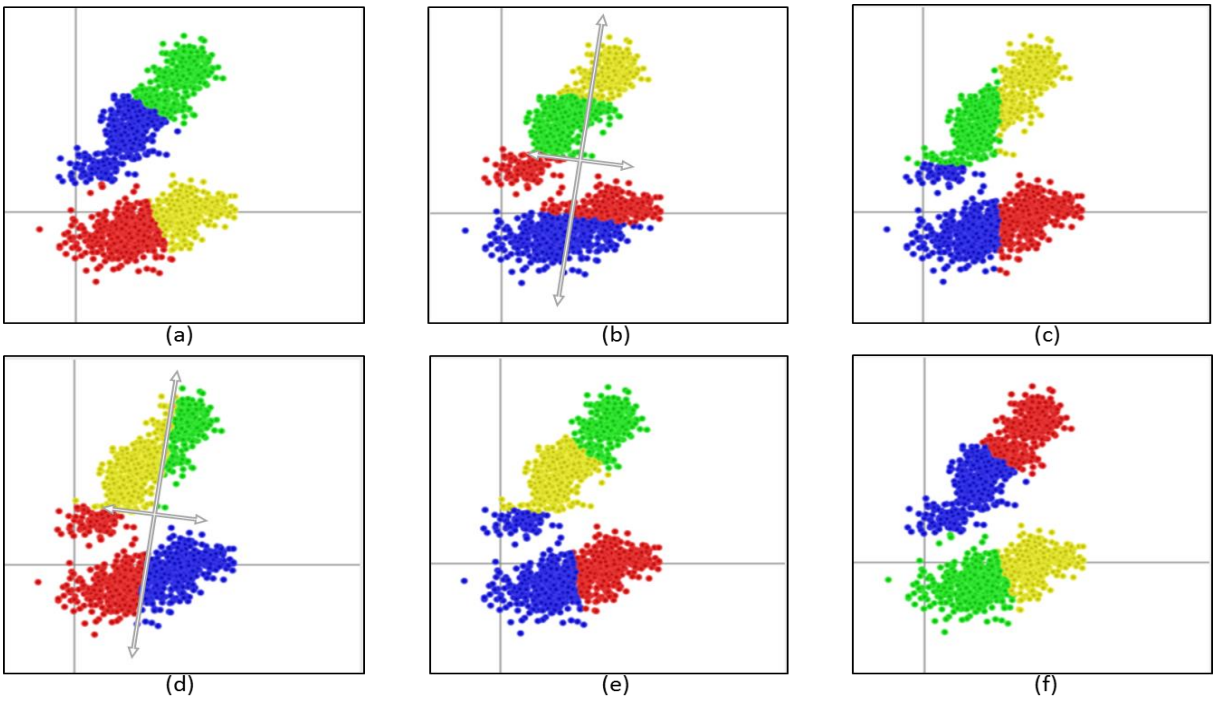

Fig. 2: A comparison of the methods (a) K-Means, (b) TC, (c) PQ, (d) OPQParametric, (e) LOPQ and our method (f) KSSQ for 2-bit quantization, obtained by running the algorithms on a 2-D toy example. For LOPQ and KSSQ there are 2 subquantizers and for K-Means obviously the number of centroids is 4. Gray arrows are the principal components.

the one with the lowest quantization error, as given in (8). This is first of all, computationally advantageous because the summation based algorithms are usually computationally expensive as shown in **TABLE 13**. This is due to the limited constraints put on the generation and selection of codevectors in order to obtain better quantization performance.

In the proposed method however, the constraints of many Cartesian product based approaches such as [12], [14], [16] are retained, which require much less computations. Thanks to the jointly optimized subspace generation approach, the assumptions of the given constraints are much more realistic for the given dataset, resulting in a quantization scheme with improved performance.

As discussed later in **Section 5.7**, the distance calculation complexity of the proposed method is comparable to other Cartesian product based approaches. Besides, one big disadvantage of the summation based approaches is that, the distance calculation is more expensive than the Cartesian product based approaches, because of the nonorthogonality between subcodebooks, as also stated in [22]. Hence using the proposed method, a faster exhaustive search is possible.

4 Experiments

The proposed approach is tested on two publicly available datasets, SIFT1M and GIST1M [12]. SIFT1M consists of 1 Million samples of 128-dimensional SIFT vectors for test, 100,000 vectors for training and 10,000 for queries. GIST1M consists of 1 Million samples of 960-dimensional GIST vectors for test, 500,000 vectors for training and 1,000 queries.

The proposed method is trained using the given training sets and exhaustive search is performed on both datasets for all queries. $K = 256$ and $\mathcal{K} = 16$ for SIFT1M, and $K = 32$ and $\mathcal{K} = 8$ for GIST1M are selected, as later justified in **Section 5**. The proposed method (**KSSQ**) is compared with the recent state-of-the-art methods from the literature such as, *Transform Coding* (**TC***) [14], *Product Quantization* (**PQ**) [12], *Cartesian K-Means/Optimized Product Quantization* (**CKM/OPQ**) [15], [16], Distance Encoded Product Quantization (**DEPQ**) [17], an exhaustive implementation of Locally Optimized Product Quantization (**E-LOPQ***) [18], *Optimized Cartesian K-Means* (**OCK**) [19], *Additive Quantization* (**AQ/APQ**) [20], *Composite Quantization* (**CQ**) [21], *Optimized Tree Quantization* (**OTQ**) [22] and *MPCA Binary Embedding* (**MPCA-E***) [28].

The results for most of the competing methods are obtained from the figures in the original publications while our own implementations of TC*, DEPQ*, E-LOPQ* and MPCA-E* are used. For DEPQ*, $K = 128$ is selected and 1 bit is allocated for distance encoding as suggested in [17]. For E-LOPQ* an exhaustive version of LOPQ is developed for fair comparison with other exhaustive methods. $K = 256$ is selected, allocating 8 bits for cluster index overhead. For MPCA-E, the multiple PCA version of the Transform Coding is selected as it provides the best retrieval performance [28]. For AQ/APQ, AQ is compared for 32-bits coding and APQ for 64-bits as suggested by the authors. *NA* indicates that the corresponding results are not presented in the original publication for the corresponding method.

TABLE 5
RECALL@R RESULTS FOR SIFT1M, 32-BIT CODES

	recall@1	recall@10	recall@100
PQ	0.052	0.230	0.595
TC*	0.057	0.197	0.519
CKM/OPQ	0.068	0.273	0.658
OCK	*NA*	0.348	0.742
AQ	0.106	0.415	**0.825**
DEPQ*	0.017	0.086	0.310
CQ	*NA*	*NA*	*NA*
E-LOPQ*	0.134	0.385	0.738
OTQ	0.093	0.368	0.793
MPCA-E*	0.124	0.404	0.784
KSSQ	**0.145**	**0.434**	0.802

TABLE 6
RECALL@R RESULTS FOR GIST1M, 32-BIT CODES

	recall@1	recall@10	recall@100
PQ	0.023	0.068	0.176
TC*	0.053	0.104	0.291
CKM/OPQ	0.054	0.142	0.396
OCK	*NA*	0.172	**0.467**
AQ	0.069	0.189	**0.467**
DEPQ*	0.025	0.092	0.323
CQ	*NA*	*NA*	*NA*
E-LOPQ*	0.049	0.131	0.362
OTQ	*NA*	*NA*	*NA*
MPCA-E*	0.054	0.149	0.345
KSSQ	**0.078**	**0.191**	0.437

TABLE 7
RECALL@R RESULTS FOR SIFT1M, 64-BIT CODES

	recall@1	recall@10	recall@100
PQ	0.224	0.599	0.924
TC*	0.205	0.535	0.877
CKM/OPQ	0.243	0.638	0.940
OCK	0.274	0.680	0.945
APQ	0.298	0.741	0.972
DEPQ*	0.139	0.432	0.806
CQ	0.288	0.716	0.967
E-LOPQ*	0.297	0.703	0.957
OTQ	0.317	0.748	0.972
MPCA-E*	0.286	0.710	0.923
KSSQ	**0.325**	**0.754**	**0.976**

TABLE 8
RECALL@R RESULTS FOR GIST1M, 64-BIT CODES

	recall@1	recall@10	recall@100
PQ	0.076	0.218	0.504
TC*	0.096	0.223	0.547
CKM/OPQ	0.118	0.334	0.715
OCK	0.130	0.358	0.720
AQ/APQ	*NA*	*NA*	*NA*
DEPQ*	0.096	0.308	0.668
CQ	0.135	0.377	0.729
E-LOPQ*	0.116	0.331	0.656
OTQ	*NA*	*NA*	*NA*
MPCA-E*	0.110	0.312	0.662
KSSQ	**0.136**	**0.396**	**0.741**

The **recall@*R*** measure is selected as the performance metric, which is the recall value for the first R retrieved samples. It is assumed that the nearest sample in the test set is the ground truth for each query, as in [16], [18]–[22], [28]. The results for **recall@1**, **recall@10** and **recall@100** arecalculated. The results obtained on SIFT1M and GIST1M datasets for 32-bit coding are presented in **TABLE 5** and **TABLE 6** and the results obtained on SIFT1M and GIST1M datasets for 64-bit coding are presented in **TABLE 7** and **TABLE 8**, respectively.

As observed from the results presented above, the proposed method outperforms all recent state-of-the-art methods for recall@1 and recall@10 scores, for all datasets and all code lengths. Only for recall@100 scores in 32-bit codes, the proposed method has been outperformed by AQ/APQ, which is computationally much more expensive as discussed in the next section.

Here it should again be emphasized that the results presented for E-LOPQ* are different from the ones presented in [18], for several reasons. First of all, since we test the methods for exhaustive search, in the exhaustive implementation of [18], all the cells have been visited for fair comparison, i.e., the number of visited cells w = K. Another reason of the difference is that, the authors did not include the localization overhead into the number of bits in [18], i.e., a sample is encoded with 10+64 = 74-bits in total, because of the coarse quantizer with $K = 1024$. For fair comparison of E-LOPQ* with other methods, this overhead is included to the total number of bits, by sparing 8 bits for coarse quantization and the remaining 56 (24) bits for subquantizers for 64-bit (32-bit) coding.

As it can be seen, formulating vector quantization as a joint optimization problem has resulted in an improved performance as expected. While [28] proves that the use of multiple PCAs improves the retrieval performance, optimizing the centroids together with affine subspace clustering as proposed in this paper has shown to outperform the state-of-the-art methods. Note that, [28] is outperformed by many state-of-the-art methods and the proposed solution brings a very significant improvement over it.

5 DISCUSSIONS

In this section, the parameter selections, quantization error, computational and storage costs of the proposed algorithm are discussed and compared with other methods.

5.1 Parameter Selection

The only parameters for the proposed method are the number of affine subspaces K and the number of selected subspaces for encoding, $\mathcal{K}$. In the experiments, it has been observed that, there is an upper limit for the number of affine subspaces K, which is brought by the training set. As the number of clusters is increased exponentially, after a certain point, empty clusters are generated, which are not assigned any samples in the update stage of the training. So the number K is

selected as the highest number for which none of the clusters is empty.

The proposed method is tested for values of K from 16 to 256 (4-bit to 8-bit) and the recall@10 scores for 64-bit coding on both SIFT1M and GIST1M dataset are presented in **Fig. 3**. As it can be seen, after $K = 32$ in GIST1M dataset, the occurrence of empty clusters brings down the performance. For SIFT1M dataset, no empty clusters are observed until K=256. Considering the number of samples in the training set, and the number of samples per cluster, cases with $K \geq 512$ are not tested. For 32-bit and 64-bit coding, $K = 256$ (8 bits) for SIFT1M and $K = 32$ (5 bits) for GIST1M are selected. Here note that, each bit spent on indexing the subspace is deduced from the bits spent for quantization. For example, a 64-bit code consists of 8 bits for subspace indexing and 56 bits for quantization.

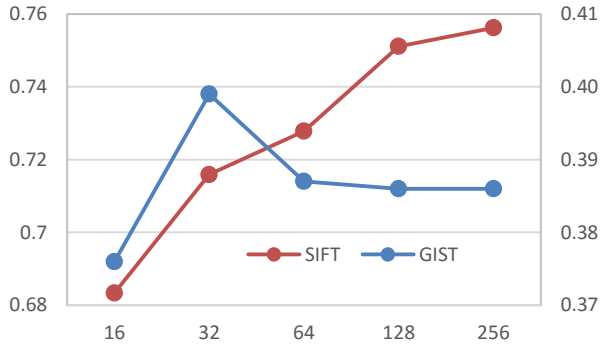

Fig. 3: recall@10 vs. K on SIFT1M and GIST1M with 64-bit codes. (left vertical axis: recall@10 for SIFT1M, right vertical axis: recall@10 for GIST1M, horizontal axis: number of subspaces K)

In order to determine $\mathcal{K}$, the quantization error on the training set for the proposed method is calculated. Starting from $\mathcal{K} = K$, $\mathcal{K}$ is decreased exponentially while keeping the increase in the quantization error less than 1% compared to the initial error where $\mathcal{K} = K$. The change in quantization error for different datasets and different code lengths is presented in **TABLE 9**. According to this, for SIFT1M dataset $\mathcal{K} = 16$ and for GIST1M $\mathcal{K} = 8$ is chosen.

TABLE 9
CHANGE IN QUANTIZATION ERROR WITH $\mathcal{K}$

	SIFT1M 64-bits	SIFT1M 32-bits	GIST1M 64-bits	GIST1M 32-bits
Encoding (K)	15253.1	26341.1	0.644	0.826
Encoding ($\mathcal{K}$)	15313.9	26406.2	0.645	0.833
Difference (%)	0.398	0.247	0.155	0.847

5.2 Breakdown of Quantization Error

As stated in **Section 3,** in the proposed method the error term in (6) is minimized by optimizing both terms. The errors obtained separately from the first and the second terms are shown in **TABLE 10**. It is compared with the case of $K = 1$, i.e., TC variant, $K = 256$, MPCA-E and OPQ using 64-bit codes. As observed from the results, the quantization error is decreased significantly, and an important part of this decrease comes from the transformation error. Comparison with MPCA-E also shows that, joint minimization of both terms results in less quantization error in total as expected. The decrease in the quantization error is presented in **Fig. 4**.

TABLE 10
BREAKDOWN OF QUANTIZATION ERROR ON SIFT1M

	$\|\|\boldsymbol{R}^T\boldsymbol{x}_i - Q(\boldsymbol{R}^T\boldsymbol{x}_i)\|\|^2$	$\left\|\boldsymbol{R}^{\perp T}\boldsymbol{x}_i\right\|^2$	**Total**
TC	16724.1	16346.1	33070.2
OPQ	22239.8	0	22239.8
MPCA-E	8988.2	7350.9	16339.1
KSSQ	5560.7	9692.4	15253.1

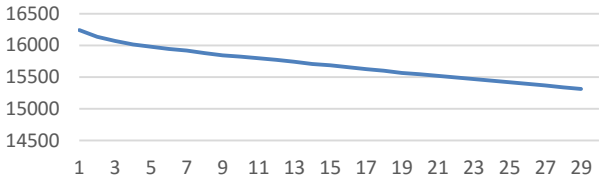

Fig. 4: The decrease in MSE_Q on SIFT1M, MSE_Q vs Iterations. (left vertical axis: MSE_Q, horizontal axis: iterations)

5.3 Comparison of Bit Allocation Methods

As mentioned in **Section 3.3**, in this paper it is proposed to use *Modified d'Hondt* method instead the greedy bit allocation procedure which was proposed in [14]. Note that the *d'Hondt* method creates allocation schemes which are similar to [14], but the *Modified d'Hondt* method emphasizes the dimensions with higher standard deviations more, creating subspaces with a reduced number of dimensions. This creates a more suitable bit allocation scheme for the proposed method, since the transformation error is already taken into account in the iterative minimization. A comparison of performances of the proposed method using the *d'Hondt* and the *Modified d'Hondt* allocation schemes is presented in **TABLE 11**. As it can be seen, KSSQ with the *Modified d'Hondt* scheme performs better than KSSQ with *d'Hondt.*

TABLE 11
RECALL@R RESULTS FOR DIFFERENT BIT ALLOCATION METHODS ON SIFT1M, 64-BIT CODES

	recall@1	**recall@10**	**recall@100**
d'Hondt	0.312	0.733	0.974
Modified d'Hondt	**0.325**	**0.754**	**0.976**

5.4 Indexing and Non-Exhaustive Search

Scalibility of binary embedding methods for billion scale datasets is usually provided by indexing methods as in [15], [18], [30], [31] . In these methods, first a coarse quantizer is applied to create an inverse file list, then, except LOPQ, the residuals of the coarse quantizer are quantized by a subquantizer. In LOPQ, separate subquantizers are trained for each coarse quantizer codeword. So a given query is compared to only a subset of the original dataset, which corresponds to the samples of the nearest cells. This provides faster, non-exhaustive search. The proposed method can be easily replaced with the subquantizer (PQ in [30], [31] or OPQ in [15]) and scale up for large scale datasets.

As an example, we follow the approach in [31] and replace the residual quantizer q_r with the proposed method and test the performance for non-exhaustive search on SIFT1B dataset [12]. SIFT1B consists of 1 billion samples with 128 dimensions. The results of the proposed method are compared with the state-of-the-art single index methods such as **IVFADC** [31], **I-OPQ** [18] and **LOPQ** [18], as presented in [18]. Similar to [18], 8192 clusters are used for indexing, and the search is performed in the nearest 64 cells. The results are presented in **TABLE 12**.

TABLE 12
RECALL@R RESULTS FOR SIFT1B, 64-BIT CODES, 13-BIT INDEXES

	recall@1	recall@10	recall@100
IVFADC	0.088	0.372	0.733
I-OPQ	0.114	0.399	0.777
LOPQ	**0.199**	**0.586**	**0.909**
I-KSSQ	0.141	0.472	0.840

As it can be seen, the proposed method performs better than other global methods, yet it is outperformed by LOPQ. Here it should be emphasized that, the local optimization requires a great amount of additional storage, around 3GB with these parameters [18], while the proposed method requires only around 20MB. Since in this paper the focus is drawn on the quality of quantization, the improvements on non-exhaustive search for KSSQ are left for future work.

5.5 Computational Cost of Training

The computational cost of the proposed training scheme can be calculated as follows. For each of the K affine subspaces, K PCAs are calculated, each with a cost of $O(D^3 + D^2N)$. Then, each sample is projected onto each affine subspace with a cost of $O(DL)$, where L is the number of dimensions after dimension reduction. To calculate the distance to the corresponding subspace, another projection with the same cost is needed. The cost of obtaining centroids in each subspace is $O(NGT_{ML})$, where T_{ML} is the number of iterations for Max-Lloyd algorithm and G is the number of bits. To update the subspaces each sample is encoded, with a cost of $O(2KDL)$ as explained below. These steps are repeated for T_{TR} iterations. $T_{ML} = 10$ and $T_{TR} = 50$ are chosen. The total training cost can thus be computed as follows:

$$O\left(T_{TR}\left(K(D^3 + ND^2) + N(2DL + GT_{ML} + 2KDL)\right)\right) \quad (14)$$

5.6 Computational Cost of Encoding

The encoding cost for the proposed method can be calculated as follows. First the first $\mathcal{K}$ subspaces among K are found with centers closest to the given sample, with a cost of $O(KD)$ ($\mathcal{K} \ll D$). For a given sample, $\mathcal{K}$ projections are performed and each projection costs $O(2DL)$, including the distance calculation to a given subspace. After projecting the sample to the subspace, approximately G comparisons are needed to find the corresponding code. So the total encoding cost is $O(2\mathcal{K}DL + KD + \mathcal{K}G)$. The cost comparison of the proposed method is presented in **TABLE 13**. As observed, KSSQ has comparable cost with respect to most of the competing methods and much less cost than the best performing competitors AQ/APQ and OTQ. For example, for 32-bit coding on GIST1M dataset, the proposed method is 4 times less costly than OTQ and 44 times less costly than AQ/APQ.

5.7 Computational Cost of the Distance Calculation

Given that projections on different subspaces are precomputed and stored for a given sample, the asymmetric distance can be calculated in less than L table lookups, i.e., one lookup for each dimension. However, if some dimensions are stored together in the lookup table, the number of lookups can be decreased to the same level as in [12], [15], [16].

5.8 Additional Storage Costs

The additional storage cost of the proposed method is comparable to that of the competing methods. For KSSQ, it is required to store the transformation matrix, the affine shift vector and the centroids for each affine subspace. The cost can be approximately expressed as $O(KDL)$. Note that this cost is independent from the size of the dataset, but depends on the number of subspaces, the dimension of the input space and the number of bits used for coding. For example, for the 960-dimensional dataset GIST1M, KSSQ requires an additional space of about 9 MB. However, for example, the storage cost of OPQ can be approximately expressed as $O(D^2 + HD)$ and for LOPQ it is $O(K(D^2 + HD))$, resulting in 9 MB for OPQ and 2 GB for LOPQ. OTQ has a storage cost of $O(D^2 + MHD)$ corresponding to 22 MB. Considering these figures, it can be said that the storage costs of KSSQ are comparable to the competing methods. A detailed comparison of the storage space requirements of KSSQ and the competing methods is given in **TABLE 13**.

6 CONCLUSION

In this study a novel vector quantization algorithm is proposed for the approximate nearest neighbor search problem. The proposed method explores the quantization centers in affine subspaces through an iterative technique, which jointly attempts to minimize the quantization error of the training samples in the learnt subspaces, while minimizing the projection error of the samples to the corresponding subspaces. The proposed method has proven to outperform the state-of-the-art-methods, with comparable computational cost and additional storage. In this paper it is also shown that, dimension reduction is an important source of quantization error, and by exploiting subspace clustering techniques the quantization error can be reduced, leading to a better quantization performance.

So far we have focused mainly on exhaustive search but an index-based non-exhaustive extension for the proposed method can be further investigated. Our approach can also be extended to labeled datasets in order to test k-nearest neighbor classification performance. These will be the topics of our future work.

TABLE 13
COMPARISON OF COMPUTATIONAL AND STORAGE COSTS

Method	Encoding Cost	Encoding Cost for Different Datasets and Code Lengths			
		SIFT1M-32	SIFT1M-64	GIST1M-32	GIST1M-64
PQ	$O(HD)$	32768	32768	245760	245760
TC	$O(DL+G)$	2592	5184	19232	38464
CKM/OPQ	$O(D^2+HD)$	49152	49152	1167360	1167360
OCK	$O(THD)$	327680	327680	2457600	2457600
AQ	$O(M^2H^2(M+\log(MH))+HD)$	14712832	79724544	14925824	79937536
APQ	$O\left(D^2+\frac{M}{4}(4^2H^2(4+\log(4H))+HD)\right)$	14729216	14761984	15847424	16093184
DEPQ	$O(D^2+HD/2)$	32768	32768	1044480	1044480
CQ	$O(3MHD)$	393216	786432	2949120	5898240
E-LOPQ	$O(\varkappa D+HD+D^2)$	81920	81920	1413120	1413120
OTQ	$O(D^2+HD+MH^2)$	311296	573440	1429504	1691648
MPCA-E	$O(2KDL+KG)$	1318912	2637824	1229824	2459648
KSSQ	$O(KD+2\mathcal{K}DL+\mathcal{K}G)$	115200	197632	338176	645632
Method	**Storage Cost**	**Storage Cost for Different Datasets and Code Lengths (MB)**			
		SIFT1M-32	SIFT1M-64	GIST1M-32	GIST1M-64
PQ	$O(HD)$	0.25	0.25	1.88	1.88
TC	$O(DL)$	0.02	0.04	0.15	0.29
CKM/OPQ	$O(D^2+HD)$	0.38	0.38	8.91	8.91
OCK	$O(D^2+2HD)$	0.63	0.63	10.78	10.78
AQ	$O(MHD)$	1.00	2.00	7.50	15.00
APQ	$O(D^2+MHD)$	1.13	2.13	14.53	22.03
DEPQ	$O(D^2+HD/2)$	0.25	0.25	7.97	7.97
CQ	$O(MHD)$	1.00	2.00	7.50	15.00
E-LOPQ	$O(\varkappa(HD+D^2))$	96.00	96.00	2280.00	2280.00
OTQ	$O(D^2+MHD)$	1.13	2.13	14.53	22.03
MPCA-E	$O(KDL)$	5.00	10.00	4.69	9.38
KSSQ	$O(KDL)$	5.00	10.00	4.69	9.38
H: number of subcodevectors		256	256	256	256
D: number of dimensions		128	128	960	960
M: number of subcodebooks		4	8	4	8
L: number of reduced dimensions (on average)		20	40	20	40
$\varkappa$: number of subspaces for LOPQ		256	256	256	256
K: number of subspaces for KSSQ and MPCA-E		256	256	32	32
$\mathcal{K}$: number of selected subspaces for encoding		16	16	8	8
G: number of bits used for encoding.		32	64	32	64
T: search depth for OCK		10	10	10	10

REFERENCES

[1] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," *Proc. thirtieth Annu. ACM Symp. Theory Comput.*, pp. 604–613, 1998.

[2] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for Similarity Search: A Survey," in *arXiv preprint*, 2014, p. :1408.2927.

[3] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-Sensitive Hashing Scheme Based on P-stable Distributions," in *SCG*, 2004, p. 253.

[4] K. Terasawa and Y. Tanaka, "Spherical LSH for Approximate Nearest Neighbor Search on Unit Hypersphere," in *WADS*, 2007, pp. 27–38.

[5] X. He, D. Cai, S. Yan, and H. Zhang, "Neighborhood Preserving Embedding," in *ICCV*, 2005.

[6] H. Jegou, M. Douze, C. Schmid, and P. Perez, "Aggregating local descriptors into a compact image representation," in *CVPR*, 2010, pp. 3304–3311.

[7] J. Heo, Y. Lee, and J. He, "Spherical hashing," in *CVPR*, 2012.

[8] A. Gordo, F. Perronnin, Y. Gong, and S. Lazebnik, "Asymmetric distances for binary embeddings.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 1, pp. 33–47, Jan. 2014.

[9] W. Dong, M. Charikar, and K. Li, "Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces," in *SIGIR*, 2008, p. 123.

[10] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[11] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognit. Lett.*, vol. 31, no. 8, pp. 651–666, 2010.

[12] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–28, Jan. 2011.

[13] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *CVPR*, 2011, pp. 817–824.

[14] J. Brandt, "Transform coding for fast approximate nearest neighbor search in high dimensions," in *CVPR*, 2010, pp. 1815–1822.

[15] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized Product Quantization.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, pp. 1–12, Dec. 2014.

[16] M. Norouzi and D. J. Fleet, "Cartesian K-Means," in *CVPR*, 2013, pp. 3017–3024.

[17] J.-P. Heo, Z. Lin, and S.-E. Yoon, "Distance Encoded Product Quantization," in *CVPR*, 2014, pp. 2139–2146.

[18] Y. Kalantidis and Y. Avrithis, "Locally Optimized Product Quantization for Approximate Nearest Neighbor Search," in *CVPR*, 2014.

[19] J. Wang, J. Wang, J. Song, X.-S. Xu, H. T. Shen, and S. Li, "Optimized Cartesian K-Means," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 180–192, Jan. 2015.

[20] A. Babenko and V. Lempitsky, "Additive Quantization for Extreme Vector Compression," in *CVPR*, 2014, pp. 931–938.

[21] T. Zhang, D. Chao, and J. Wang, "Composite Quantization for Approximate Nearest Neighbor Search," in *ICML*, 2014.

[22] A. Babenko and V. Lempitsky, "Tree Quantization for Large-Scale Similarity Search and Classification," in *CVPR*, 2015.

[23] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Trans. Inf. Theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[24] N. Kambhatla and T. K. Leen, "Dimension Reduction by Local Principal Component Analysis," *Neural Comput.*, vol. 9, no. 7, pp. 1493–1516, Oct. 1997.

[25] V. Gassenbauer, J. Křivánek, K. Bouatouch, C. Bouville, and M. Ribardière, "Improving Performance and Accuracy of Local PCA," *Comput. Graph. Forum*, vol. 30, no. 7, pp. 1903–1910, Sep. 2011.

[26] P. Agarwal and N. Mustafa, "k-Means Projective Clustering," in *SIGMOD*, 2004, pp. 155–165.

[27] C. M. Bishop, "Bayesian PCA," in *NIPS*, 1999, vol. 11, pp. 382–388.

[28] E. C. Ozan, S. Kiranyaz, and M. Gabbouj, "M-PCA Binary Embedding For Approximate Nearest Neighbor Search," in *BigDataSE*, 2015.

[29] M. Gallagher, "Proportionality, disproportionality and electoral systems," *Electoral Studies*, vol. 10. pp. 33–51, 1991.

[30] A. Babenko and V. Lempitsky, "The inverted multi-index," in *CVPR*, 2012, vol. 14, no. 1–3, pp. 3069–3076.

[31] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: Re-rank with source coding," *ICASSP*, no. 3, pp. 861–864, 2011.

Ezgi Can Ozan received his BS degree in Electrical and Electronics Department at Middle East Technical University, Ankara, Turkey, in 2007 and MS degree in Signal Processing from the same University, in 2011. He is currently a Ph.D candidate and working as a researcher at Tampere University of Technology, Tampere, Finland. His areas of interest include large-scale multimedia search, pattern recognition, machine learning, and computer vision.

Serkan Kiranyaz received his BS degree in Electrical and Electronics Department at Bilkent University, Ankara, Turkey, in 1994 and MS degree in Signal and Video Processing from the same University, in 1996. He worked as a Senior Researcher in Nokia Research Center and later in Nokia Mobile Phones, Tampere, Finland. He received his PhD degree from Tampere University of Technology; Institute of Signal Processing in 2005 and his Docency at 2007 respectively. He is currently working as a Professor in Electrical Engineering Department of Qatar University. Prof. Kiranyaz published 2 books, more than 35 journal papers on several IEEE Transactions and some other high impact journals and 80+ papers in international conferences. His recent publication, "Automatic Object Segmentation by Quantum Cuts" won the IBM

Best Paper Award in ICPR'14. Prof. Kiranyaz is a senior of IEEE.

Moncef Gabbouj received his BS degree in electrical engineering in 1985 from Oklahoma State University, Stillwater, and his MS and PhD degrees in electrical engineering from Purdue University, West Lafayette, Indiana, in 1986 and 1989, respectively. Dr. Gabbouj is a Professor of Signal Processing at the Department of Signal Processing, Tampere University of Technology, Tampere, Finland. He was Academy of Finland Professor during 2011-2015. He held several visiting professorships at different universities. His research interests include multimedia content-based analysis, indexing and retrieval, machine learning, nonlinear signal and image processing and analysis, voice conversion, and video processing and coding. Dr. Gabbouj is a Fellow of the IEEE and member of the Academia Europaea and the Finnish Academy of Science and Letters. He is the past Chairman of the IEEE CAS TC on DSP and committee member of the IEEE Fourier Award for Signal Processing. He served as Distinguished Lecturer for the IEEE CASS. He served as associate editor and guest editor of many IEEE, and international journals. Dr. Gabbouj was the recipient of the 2015 TUT Foundation Grand Award, the 2012 Nokia Foundation Visiting Professor Award, the 2005 Nokia Foundation Recognition Award, and several Best Paper Awards. He published over 650 publications and supervised 40 doctoral theses.

III

SELF-ORGANIZED BINARY ENCODING FOR APPROXIMATE NEAREST NEIGHBOR SEARCH

by

E.C.Ozan, S. Kiranyaz, M. Gabbouj & X. Hu, August 2016

24th European Signal Processing Conference (EUSIPCO), Budapest, 2016, pp. 1103-1107.

Self-Organizing Binary Encoding for Approximate Nearest Neighbor Search

Ezgi Can Ozan
Signal Processing Department
Tampere University of Technology
Tampere, Finland
ezgi.ozan@tut.fi

Serkan Kiranyaz
Electrical Engineering Department,
College of Engineering
Qatar University
Qatar
mkiranyaz@qu.edu.qa

Moncef Gabbouj
Signal Processing Department
Tampere University of Technology
Tampere, Finland
moncef.gabbouj@tut.fi

Xiaohua Hu
College of Computing and Informatics
Drexel University
Philadelphia, PA, USA
xh29@drexel.edu

***Abstract*—Approximate Nearest Neighbor (ANN) search for indexing and retrieval has become very popular with the recent growth of the databases in both size and dimension. In this paper, we propose a novel method for fast approximate distance calculation among the compressed samples. Inspiring from Kohonen's self-organizing maps, we propose a structured hierarchical quantization scheme in order to compress database samples in a more efficient way. Moreover, we introduce an error correction stage for encoding, which further improves the performance of the proposed method. The results on publicly available benchmark datasets demonstrate that the proposed method outperforms many well-known methods with comparable computational cost and storage space.**

Keywords- Approximate Nearest Neighbor Search; Quantization; Binary Feature Synthesis; Vector Compression, Self-Organizing Maps.

I. INTRODUCTION

The increase in the amount of daily generated data has become a significant research problem in recent years. As the size and dimension of the generated data grew, traditional methods began to fail to produce satisfactory results; therefore, new solutions specific for very large-scale datasets are desired. An important approach to handle the aforementioned problem is to develop approximate solutions. Approximate Nearest Neighbor Search (ANN) has become a noteworthy research topic in recent years for indexing and retrieval in very large-scale datasets [1]. ANN aims to compress the dataset samples as binary strings, then estimate the distance between the dataset samples and novel queries in a computationally efficient way.

Methods in the literature for ANN can be split into two main branches: Hashing and Vector Quantization (VQ). Hashing methods aim to approximate the distance between samples by calculating the Hamming distance between binary strings. Calculation of Hamming distance is very fast but since distances are integer values, the accuracy is inferior [2]–[6]. VQ based methods aim to approximate the distance between samples by using look-up tables of pre-calculated distances between learned codevectors and have proven to outperform Hashing based methods in terms of retrieval accuracy [7]–[11].

In this paper, we propose a novel VQ based binary encoding method, which performs ANN search on very large-scale datasets in a computationally efficient way. We inspire from Kohonen's self-organizing maps [12] and propose a hierarchical structure of several layers of quantization. We compare our method with well-known methods from the literature and our method outperforms those methods with comparable computational and storage costs.

The rest of the paper is organized as follows: In Section II, the problem formulation is defined and related works in the literature are introduced. In Section III, the proposed method is explained in detail. Section IV represents the findings of experiments on large scale datasets and finally in Section V the paper is concluded.

II. PROBLEM FORMULATION AND RELATED WORK

In this section, we first define VQ as an optimization problem. For a quantizer Q, given a set of D dimensional N input vectors $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\} \in \mathbb{R}^{D \times N}$, the mean squared quantization error MSE_Q is defined as in (1).

$$MSE_Q = \frac{1}{N}\sum_{i}^{N} \|\boldsymbol{x}_i - Q(\boldsymbol{x}_i)\|^2 \tag{1}$$

Among the K codevectors in the codebook matrix $\boldsymbol{C} \in \mathbb{R}^{D \times K}$ of Q, the suitable one is selected by a binary selection vector $\boldsymbol{b}_i \in \{0,1\}^K$ is the binary selection vector, with $\|\boldsymbol{b}_i\| = 1$.

$$Q(\boldsymbol{x}_i) = \boldsymbol{C}\boldsymbol{b}_i \tag{2}$$

In ANN adaptations of VQ, the number of codevectors should be much greater than the traditional VQ. This is obtained by using several codebooks for quantization [1]. By using M different codebooks, the number of codevectors can be increased from K to K^M.

Product Quantization (PQ) by Jégou *et al.* [7] is among the first examples of VQ for ANN. The authors propose to divide the feature space into M subspaces, therefore, the optimization

is split into M problems and solved in each subspace separately. However, the division of the feature space requires the assumption of statistical independency among the subspaces, which is not realistic in practice. Several improvements on PQ have been proposed in order to overcome this drawback such as [8], [9]. The optimization problem can be formulated as given in (3).

$$MSE_Q^{(PQ_1)} = \min_{\{\boldsymbol{C}_1\},\{\boldsymbol{B}_1\}} \frac{1}{N}\sum_{i=1}^{N}\|\boldsymbol{x}_i - \boldsymbol{C}_1\boldsymbol{b}_{1_i}\|^2$$
$$\vdots \qquad (3)$$
$$MSE_Q^{(PQ_M)} = \min_{\{\boldsymbol{C}_M\},\{\boldsymbol{B}_M\}} \frac{1}{N}\sum_{i=1}^{N}\|\boldsymbol{x}_i - \boldsymbol{C}_M\boldsymbol{b}_{M_i}\|^2$$

A. *Residual Vector Quantization*

Residual Vector Quantization (RVQ) by Chen *et al.* [11] is another significant example for VQ for ANN, which follows a totally different approach than PQ. The authors propose to create hierarchical layers of quantization, where in each layer the residuals of the previous layers are quantized. So the optimization problem is again split into M problems and solved separately. RVQ can be formulated as given in (4).

$$MSE_Q^{(RVQ_1)} = \min_{\{\boldsymbol{C}_1\},\{\boldsymbol{B}_1\}} \frac{1}{N}\sum_{i=1}^{N}\left\|\left(\boldsymbol{x}_i - \sum_{m=1}^{1}\boldsymbol{C}_m\boldsymbol{b}_{m_i}\right)\right\|^2$$
$$\vdots \qquad (4)$$
$$MSE_Q^{(RVQ_M)} = \min_{\{\boldsymbol{C}_M\},\{\boldsymbol{B}_M\}} \frac{1}{N}\sum_{i=1}^{N}\left\|\left(\boldsymbol{x}_i - \sum_{m=1}^{M}\boldsymbol{C}_m\boldsymbol{b}_{m_i}\right)\right\|^2$$

As it can be seen in the equation above, even the problem is split into M layers, the optimization in each layer depends on the solutions of the previous layers. In other words, the selection of codevectors in each layer highly affects the selection of the codevectors in the next layers.

In RVQ, each problem is solved using the K-Means algorithm. K-Means algorithm converges to a local minima in each layer, but when upper layers overfit to the data, the contribution of lower layers decreases, leading to inferior codebooks. In this paper, we introduce a structure for each codebook, to prevent overfitting in higher layers and obtain better overall quantization.

III. The Proposed Method

The proposed method aims to improve the training of codebooks by imposing a structure which is inspired from the self-organizing maps in order to prevent overfitting.

A. *Self-Organizing Maps*

Self-organizing maps (SOM) can be considered as neural networks, which spatially order the responses of neurons. It is inspired from the biological fact that, in brain, the cells which are together generally respond together. In this iterative algorithm, the neurons are connected to each other by a predefined topology. When one neuron is updated, the neurons within the predefined neighborhood are updated as well as a result of the imposed structure.

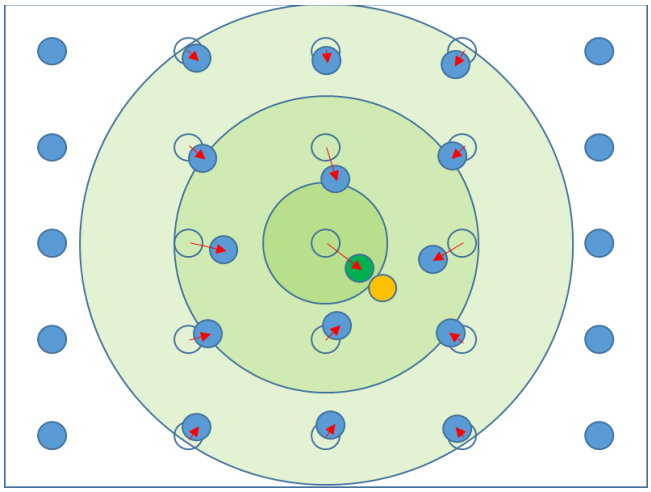

Figure 1. Illustration of SOM neuron update.

Training of SOM is a competitive learning process, i.e., for each input vector, the best matching neuron is updated to match even more closely to the corresponding vector. For an input vector $\boldsymbol{x}$, the best matching neuron is defined as given in (5).

$$\mathrm{k}^* = \underset{k}{\operatorname{argmin}}(\|\boldsymbol{x} - \boldsymbol{c}_k\|^2) \qquad (5)$$

As no closed form solution is available for the problem above, iterative approximations are used. Using the stochastic gradient descent approach, the update equation can be formulated as given in (6).

$$\boldsymbol{c}_k(t+1) = \boldsymbol{c}_k(t) - \gamma(t)\nabla_{\boldsymbol{c}_k}(\|\boldsymbol{x} - \boldsymbol{c}_k\|^2) \qquad (6)$$

where $\nabla_{\boldsymbol{c}_k}$ is the gradient operation and $\gamma(t)$ is the learning rate. The corresponding gradient can be formulated as given in (7).

$$\nabla_{\boldsymbol{c}_k}(\|\boldsymbol{x} - \boldsymbol{c}_k\|_2^2) = 2(\boldsymbol{c}_k - \boldsymbol{x}) \qquad (7)$$

Using (5), (6) and (7), the update equation for the self-organizing maps with the imposed topology can be formulated as given in (8), where $\boldsymbol{c}_{k^*}$ is the winner neuron and $\mathcal{N}_{\boldsymbol{c}_{k^*}}$ represents the set of neighboring neurons, as defined by the topology [12].

$$\boldsymbol{c}_k(t+1) = \begin{cases} \boldsymbol{c}_k(t) - \gamma(t)(\boldsymbol{c}_k - \boldsymbol{x}) & if\ k \in \mathcal{N}_{\boldsymbol{c}_{k^*}} \\ \boldsymbol{c}_k(t) & else \end{cases} \qquad (8)$$

The neighborhood can be defined by a Gaussian kernel function as given in (9) and (10), where $\mathrm{h}_{(k,k^*)}(t)$ is the kernel function, r_k is the position vector of the k^{th} neuron, $h_0(t)$ and $\sigma(t)$ are two suitable functions which decrease with time.

$$\boldsymbol{c}_k(t+1) = \boldsymbol{c}_k(t) - h_{(k,k^*)}(t)(\boldsymbol{c}_k - \boldsymbol{x}) \qquad (9)$$

$$h_{(k,k^*)}(t) = h_0(t)e^{-\|r_k - r_{k^*}\|^2/\sigma(t)^2} \qquad (10)$$

Alternatively, the distance between neurons can be used to define the neighborhood relations [12]. In our approach we follow this method in order to discard the parameter $\sigma(t)$, which is dataset dependent. We define a number of nearest neurons to

This work is funded by the TEKES project "Big Data Analysis" 32.20.40.2/15 and Tieto Finland Oy.

TABLE I. PSEUDO-CODE FOR MULTI-DIMENSIONAL TOPOLOGY

Given: $\boldsymbol{X}$: the set of input samples K: total number of neurons
k_l: The number of centroids assigned to dimension l. $\boldsymbol{V}$: vector of standard deviations, V_l: The standard deviation for dimension l. • Apply PCA on $\boldsymbol{X}$ and obtain $\boldsymbol{V}$ • while $\prod 2^{k_l} < K$, ○ $l^* = \max_l V_l$, ○ $V_{l^*} = V_{l^*}/2$ ○ $k_{l^*} = k_{l^*} + 1$ • Obtain 2^{k_l} centroids on each dimension l • Concatenate all permutations of obtained centroids to initialize the positions of neurons.

the winner neuron as the neighbor set and decrease this number exponentially with the iterations.

The neuron update process for SOM is illustrated in Figure 1. As it can be seen, the winner neuron (green disc) and all the neighboring neurons around it (blue discs) are updated to match even better (to move closer) to the given sample (yellow disc).

B. *Multi-Dimensional Topology*

Generally self-organizing maps define the network topology in a 2D form [13]. Here in this paper, we propose a multidimensional initialization for the imposed structure based on the Principal Component Analysis (PCA). As the number of neurons are predefined in SOM, we aim to distribute these neurons uniformly within the feature space.

The proposed algorithm starts with transforming the training data using PCA, in order to obtain the principal components and the corresponding standard deviations. On each principal component, we perform 1D clustering using Max-Lloyd quantization, which is very similar to K-Means [14]. To achieve the uniform distribution the number of centroids for each dimension is kept directly proportional to the standard deviation. After the centroids are distributed uniformly in the transform space, concatenating the centroids of each dimension for all the permutations, initial positions of the neurons are obtained.

Here note that, the number of all the permutations is, $\lambda_1 \times \lambda_2 \times ... \times \lambda_L$, where λ_l is the number of centroids on the l^{th} dimension and L is the number of centroid appointed dimensions. In order to ensure that the total number of neurons $K = \prod_1^L \lambda_l$, we select K to be a power of 2, and we also distribute the centroids to the dimensions with the powers of 2. The algorithm that is used for the distribution of neurons among the feature space is given in TABLE I.

In RVQ's top down hierarchical scheme [11], each codevector in a lower layer is a residual of the corresponding upper layer. In other words, we can consider the lower layer codevectors as replicated and put around each and every one of the higher layer codevectors. This is illustrated in Figure 2. As a result of this scheme, the positions of lower layer codevectors

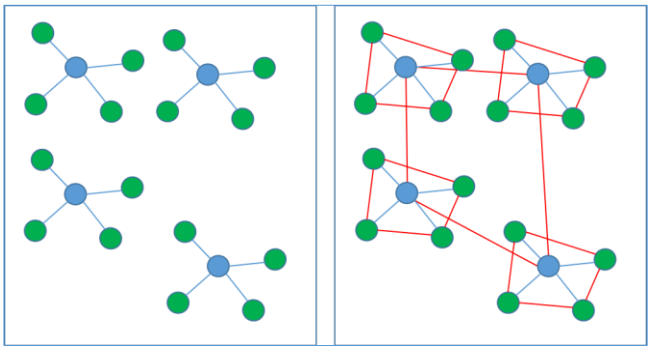

Figure 2. RVQ connections (left) vs SOBE connections (right).

are highly dependent on the positions of higher layer codevectors in the feature space. While this method has proved to be a very efficient way of exponentially increasing the total number of codevectors, the connection between the higher and lower layers also helps the distribution of codevectors among the space [11].

However, in the proposed method, we aim to add another connection among the codevectors which are in the same layer, by using SOMs. It is an intuitive expectation that, the connections between the codevectors of the same and different layers will improve the quality of training, leading to lower quantization error and better retrieval results.

C. *Self-Organized Binary Embedding (**SOBE**)*

As mentioned earlier, the proposed method uses SOMs in a top-down hierarchical order to perform quantization on vectors. At each layer, codevectors are calculated by training a SOM and then the best matching codevector (neuron weight) is subtracted from each sample in order to obtain the residuals similar to [11]. The residuals are carried to the next layer of quantization. Again for this layer, a SOM is trained and the best matching codevector of the new SOM is subtracted from the residuals in order to obtain the next set of residuals and so on. This is repeated until the final layer is reached.

Each layer of SOM is trained on the residuals of the previous layer, hence decreases the quantization error. Thanks to the initial structure of SOM, overfitting is evaded at each layer. Preventing the overfitting on upper layers, we can obtain a better quantization scheme, which utilizes the lower layers more efficiently.

D. *Encoding Correction*

Since our method uses the same hierarchical structure with RVQ, it is possible to propose a similar encoding method. The hierarchical encoding in RVQ takes place as follows: First the winner codevector for a layer is obtained, then the corresponding residual is calculated and passed on to the next layer. This is repeated for each layer.

In order to further improve the quality of encoding, we propose an additional encoding correction step on top of the hierarchical encoding approach mentioned above. For each layer, all the winner codevectors except the codevector of a given layer are subtracted from the given sample and the winner codevector for the given layer is recalculated. If the distance

TABLE II. RESULTS FOR SIFT1M, 32-BIT CODES

	recall@1	**recall@10**	**recall@100**
PQ	0.052	0.230	0.595
CKM	0.068	0.273	0.658
OCKM	NA	**0.348**	**0.742**
RVQ	NA	NA	NA
SOBE	**0.010**	**0.348**	0.731

TABLE III. RESULTS FOR SIFT1M, 64-BIT CODES

	recall@1	**recall@10**	**recall@100**
PQ	0.224	0.599	0.924
CKM	0.243	0.638	0.940
OCKM	0.273	0.680	0.945
RVQ	0.257	0.653	0.946
SOBE	**0.282**	**0.701**	**0.962**

TABLE IV. RESULTS FOR GIST1M, 32-BIT CODES

	recall@1	**recall@10**	**recall@100**
PQ	0.023	0.068	0.176
CKM	0.054	0.142	0.396
OCKM	NA	0.172	**0.468**
RVQ	NA	NA	NA
SOBE	**0.064**	**0.189**	0.403

TABLE V. RESULTS FOR GIST1M, 64-BIT CODES

	recall@1	**recall@10**	**recall@100**
PQ	0.076	0.218	0.504
CKM	0.118	0.334	0.715
OCKM	0.130	0.358	**0.719**
RVQ	0.113	0.325	0.676
SOBE	**0.136**	**0.360**	0.705

between the given sample and the corrected encoding is less than the initial encoding, then the corrected one is proposed as the final encoding.

IV. EXPERIMENTS

We test the performance of the proposed method in two publicly available datasets, SIFT1M and GIST1M [7]. Both datasets consist of 1 Million samples. SIFT1M consists of 128-dimensional SIFT vectors and GIST1M consists of 960-dimensional GIST vectors. We use the given training sets for training the codebooks and perform exhaustive search with the given queries. We use the parameters of $K = 256$, $M = 8$ for 64-bits and $M = 4$ for 32-bits coding.

A. ANN Search

The performance of the proposed method, **SOBE**, is compared against *Product Quantization* (**PQ**) [7], *Cartesian K-Means* (**CKM**) [9], *Optimized Cartesian K-Means* (**OCKM**) [10] and *Residual Vector Quantization* (**RVQ**) [11]. We also use **recall@R** performance metric, which is the average recall for the nearest neighbor in the first **r** retrievals, similar to the other methods in literature [1]. The results of the competing methods are taken from the provided figures in the original publications. The 32-bit and 64-bit encoding performances for SIFT1M dataset are given in TABLE II. and in TABLE III. . For GIST1M, the results are presented in TABLE IV. and in TABLE V. respectively.

As it can be seen in the results from the corresponding tables, the proposed method outperforms the compared methods for **recall@1** and **recall@10** in all tests. For **recall@100** the proposed method is outperformed by **OCKM** except SIFT1M 64-Bit test. The tests prove that the proposed enhancements on training and encoding have improved the nearest neighbor search performance.

In TABLE VI. the effect of encoding correction improvement has been shown in comparison with standard encoding on Sift1M dataset using 64-bit codes. As it can be seen, the proposed encoding method significantly improves the performance.

TABLE VI. EFFECT OF ENCODING CORRECTION ON SIFT1M, 64-BIT CODES

	recall@1	**recall@10**	**recall@100**
Standard	0.275	0.691	0.961
Corrected	**0.282**	**0.701**	**0.962**

B. Computational and Storage Costs

The computational cost of encoding of the proposed algorithm can be analyzed as follows: There are M hierarchical layers and for each layer, a given sample is compared with K codevectors of D dimensions each. The correction stage costs almost the same as encoding stage, so in total the cost of encoding is, $O(2MKD)$.

As it can be seen in TABLE VII. , the computational cost of encoding of the proposed method is higher than RVQ, PQ and CKM and comparable to OCKM, whereas the proposed method performs significantly better. In terms of storage costs, our method has the same requirements with RVQ, which is comparable to the storage requirements of the compared methods. The detailed comparison of storage costs are represented in TABLE VIII.

TABLE VII. COMPUTATIONAL COSTS FOR ENCODING

PQ	$O(KD)$
CKM	$O(D^2 + KD)$
OCKM	$O(10KD)$
RVQ	$O(MKD)$
SOBE	$O(2MKD)$

TABLE VIII. STORAGE COSTS

PQ	$O(KD)$
CKM	$O(D^2 + KD)$
OCKM	$O(D^2 + 2KD)$
RVQ	$O(MKD)$
SOBE	$O(MKD)$

V. CONCLUSIONS

In this paper, we propose a novel hierarchical vector quantization method, which splits the quantization problem into hierarchical layers and trains a SOM at each layer. The obtained residuals are quantized in the next layer, providing reduction in quantization error and efficient encoding. Moreover, we propose a code correction step in order to decrease the quantization error further. The experimental results show that our method performs better than the compared methods with comparable encoding and storage costs. As a future work we aim to investigate the performance of our method on billion scale datasets, and further test our algorithm on class labeled datasets using k-NN classification.

REFERENCES

[1] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for Similarity Search: A Survey," in *arXiv preprint*, 2014, p. :1408.2927.

[2] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for scalable image retrieval," in *CVPR*, 2010, pp. 3424–3431.

[3] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *CVPR*, 2011, pp. 817–824.

[4] P. Li, M. Wang, and J. Cheng, "Spectral hashing with semantically consistent graph for image indexing," *IEEE Trans. Multimed.*, vol. 15, no. 1, pp. 141–152, 2013.

[5] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognit. Lett.*, vol. 31, no. 11, pp. 1348–1358, Aug. 2010.

[6] A. Gordo, F. Perronnin, Y. Gong, and S. Lazebnik, "Asymmetric distances for binary embeddings.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 1, pp. 33–47, Jan. 2014.

[7] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–28, Jan. 2011.

[8] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized Product Quantization.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, pp. 1–12, Dec. 2014.

[9] M. Norouzi and D. J. Fleet, "Cartesian K-Means," in *CVPR*, 2013, pp. 3017–3024.

[10] J. Wang, J. Wang, J. Song, X.-S. Xu, H. T. Shen, and S. Li, "Optimized Cartesian K-Means," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 180–192, Jan. 2015.

[11] Y. Chen, T. Guan, and C. Wang, "Approximate nearest neighbor search by residual vector quantization," *Sensors*, vol. 10, no. 12, pp. 11259–11273, 2010.

[12] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, no. 1–3, pp. 1–6, 1998.

[13] M. Attik and L. Bougrain, "Self-organizing Map Initialization," in *ICANN*, 2005, pp. 357–362.

[14] S. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 129–137, 1982.

IV

JOINT K-MEANS QUANTIZATION FOR APPROXIMATE NEAREST NEIGHBOR SEARCH

by

E. C. Ozan, S. Kiranyaz & M. Gabbouj, 2016

23rd International Conference on Pattern Recognition (ICPR), Cancun, Mexico, 2016, pp. 3645-3649.

Joint K-Means Quantization for Approximate Nearest Neighbor Search

Ezgi Can Ozan
Signal Processing Department
Tampere University of Technology
Tampere, Finland
ezgi.ozan@tut.fi

Serkan Kiranyaz
Electrical Engineering Department, College of Engineering,
Qatar University ,Qatar
mkiranyaz@qu.edu.qa

Moncef Gabbouj
Signal Processing Department
Tampere University of Technology
Tampere, Finland
moncef.gabbouj@tut.fi

Abstract

Recently, Approximate Nearest Neighbor (ANN) Search has become a very popular approach for similarity search on large-scale datasets. In this paper, we propose a novel vector quantization method for ANN, which introduces a joint multi-layer K-Means clustering solution for determination of the codebooks. The performance of the proposed method is improved further by a joint encoding scheme. Experimental results verify the success of the proposed algorithm as it outperforms the state-of-the-art methods.

1 Introduction

With the growth of the cardinality and the size of current datasets, traditional methods for similarity search have begun to fail in providing satisfactory performances. In spite of the improvements in the computer hardware technology, the growth of the datasets is too fast that these improvements alone cannot catch up. One leg of this problem consists of the required storage space. As the datasets grow bigger and bigger both in number and cardinality, it gets harder to fit them into the RAM of a single computer. This brings the necessity of either dividing the datasets into multiple computers or performing search sequentially on smaller subsets of the entire dataset. Both solutions add significant computational overhead. The second leg of the problem is the computational complexity, as similarity search distance metrics such as Euclidean or Cosine distances are computationally very expensive to calculate on samples with high cardinality. This cost is also linear with the number of samples for exhaustive search, which makes it even harder for large-scale datasets.

In order to propose a solution to the aforementioned problem, approximate methods are investigated. The most popular approach is to encode the vectors using binary strings and approximate the distance between these encodings. This approach is called "hashing" [1]. These methods generally aim to find hyperplanes which divide the feature space into balanced partitions, to maximize the entropy and minimize the mutual information for binary encoding [2]–[7]. The main advantage of the hashing methods is their computational efficiency in distance calculations, as the Hamming distance calculation is performed by XOR operations followed by a '*popcount*' [8]. However, the Hamming distance is very limited in performance as the obtained distances are integers. The proposed alternative "weighted Hamming distance" brings a solution to this problem [9] by adjusting the contribution of each bit to the distance calculation. The performance is increased, but the distance calculations are performed using look-up tables instead of XOR operations [7]–[11].

Similar to weighted Hamming distances, another look-up based distance calculation method is proposed by Jègou *et al.* which opens a new era for binary embedding methods, called the Product Quantization (PQ) [8]. In this method, the feature space is divided into several subspaces and for each subspace, a separate quantizer is trained on each subspace. The codevectors obtained from the subspaces are then concatenated in order to form the final quantized vector. With this quantization based binary encoding approach, PQ significantly outperforms the Hamming distance based methods [8]. Several improvements on PQ have been proposed such as [12]–[15] carrying the performance to the state-of-the-art.

Another quantization based approach for binary encoding of feature vectors is proposed by Chen *et al.* called Residual Vector Quantization (RVQ) [16]. In this approach, the authors propose to perform quantization on the feature space using several residual layers, i.e., each layer of quantization uses the residuals of the previous layer. RVQ also outperforms the Hamming distance based methods with a significant margin [16].

The quantization based approaches mentioned above can be observed as an approximate representation of a sample vector by addition of several codevectors. PQ and variations introduce orthogonality on subvectors where as RVQ imposes a hierarchical layer structure. Recent methods such as [17]–[20] aim to find a more generic representation for the codebooks by relaxing the aforementioned constraints. The obtained results outperforms the former ones, but the search space for encoding expands as a result of the relaxed constraints, hence the computational complexity for encoding is increased.

In this paper, we propose a well-constrained method, which follows the layered structure of RVQ, but improves the performance significantly thanks to a joint approach. Our method is shown to outperform the state-of-the-art methods, by experiments conducted on publicly available benchmark datasets. The rest of the paper is organized as follows: In **Section 2**, the problem definition is presented together with related works from the literature. In **Section 3**, the proposed

method is explained in detail. **Section 4** presents the experimental results and finally the paper is concluded in **Section 5**.

2 Problem Definition and Related Work

2.1 Vector Quantization Problem

Given a set $\boldsymbol{X}$ of N vectors with D dimensions $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\} \in \mathbb{R}^{D \times N}$, the Vector Quantization (VQ) problem can be expressed as the minimization of the *Mean Squared Quantization Error*, MSE_Q, as defined in (1), where $Q(\boldsymbol{x}_i)$ is the corresponding quantized vector for given the sample vector $\boldsymbol{x}_i$.

$$MSE_Q = \frac{1}{N}\sum_{i}^{N} \|\boldsymbol{x}_i - Q(\boldsymbol{x}_i)\|^2 \qquad (1)$$

The main difference between traditional VQ algorithms and the VQ approaches proposed for ANN is that, the latter consists of M different codebooks and each codebook has K codevectors, i.e., $\boldsymbol{C}_m = \{\boldsymbol{c}_{m_1} \dots \boldsymbol{c}_{m_K}\} \in \mathbb{R}^{D \times K}$, so that the total number of codevectors is increased from K to K^M. These different codebooks can either be obtained in different subspaces as in PQ [8] or in the same space as in RVQ [16]. The quantized vector output for a quantizer can be formulated as given in (2), where $\boldsymbol{b}_i \in \{0,1\}^K$ is the binary selection vector, with $\|\boldsymbol{b}_i\| = 1$.

$$Q(\boldsymbol{x}_i) = \sum_{m=1}^{M} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \qquad (2)$$

Combining (1) and (2), the optimization problem is transformed into the following:

$$MSE_Q{}^* = \min_{\{\boldsymbol{C}_m\},\{\boldsymbol{B}_m\}} \frac{1}{N}\sum_{i}^{N} \left\| \boldsymbol{x}_i - \sum_{m=1}^{M} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \right\|^2 \qquad (3)$$

As it can be seen in (3), a closed form solution for the given minimization does not exist, hence approximate solutions are investigated. In the literature, many iterative approximations with different constraints and heuristics can be found [16]–[20]. In this section of the paper, we focus on RVQ [16] as the proposed method is mainly founded on its layered structure.

2.2 Residual Vector Quantization

Residual Quantization is a well-studied quantization method in the past [21]. The aim is to add several layers of quantizers, which operate on the residuals of the previous layers, in order to decrease the quantization error. Recent adaptations of residual quantization to VQ for ANN search have been proposed, as RVQ [16] being the first method which adapts this hierarchical structure of residual quantization for ANN. The optimization problem for RVQ can be formulated as given in (4).

$$\begin{aligned} MSE_Q^{(RVQ_1)} &= \min_{\{\boldsymbol{C}_1\},\{\boldsymbol{B}_1\}} \frac{1}{N}\sum_{i=1}^{N} \left\| \left(\boldsymbol{x}_i - \sum_{m=1}^{1} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \right) \right\|^2 \\ &\vdots \\ MSE_Q^{(RVQ_M)} &= \min_{\{\boldsymbol{C}_M\},\{\boldsymbol{B}_M\}} \frac{1}{N}\sum_{i=1}^{N} \left\| \left(\boldsymbol{x}_i - \sum_{m=1}^{M} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \right) \right\|^2 \end{aligned} \qquad (4)$$

As shown in (4), the optimization problem in (3) is divided into M sub-problems, which are solved in an order. Each layer takes the codebooks of the previous layers as fixed, hence the optimization problem is solved only for the codebook of the corresponding layer. RVQ starts with quantization of the first layer using K-Means. After each vector is quantized to its nearest codevector, the residuals are calculated and transferred to the next layer, where they will be quantized again using K-Means. This operation continues repeatedly for all layers.

The same approach is also followed in the encoding stage. When a novel sample is given, first it is encoded in the upmost layer by the index of the nearest codevector, and the corresponding residual is calculated. This residual is transferred to the next layer and this operation is repeated for all layers.

When examined in detail, RVQ has a major flow in its solution to the optimization problem defined in (4). In RVQ, each layer is trained separately, keeping the previous layers fixed. Since each layer only aims to minimize the quantization error of its own, it does not take neither upper nor lower layers into account. This results in generation of inferior codebooks, as the interlayer correlations are neglected. Similarly, encoding is performed separately in each layer, leading to a suboptimal encoding scheme.

3 Proposed Method

In this study, we aim to fix the aforementioned flow of RVQ by introducing a joint training and encoding scheme. As mentioned above, in RVQ the codebooks are trained separately, keeping the codebooks of the previous layers fixed, hence neglecting the interlayer correlations. In order to solve this problem, we propose an iterative joint training scheme which we call the *Joint K-Means*.

3.1 Joint K-Means Training

K-Means algorithm is a well-known clustering method, which aims to minimize the quantization error iteratively [22]. The iterations of K-Means follow an *"Expectation-Maximization"* approach. In the *"Expectation"* stage of iterations, each vector is appointed to its nearest codevector, in other words each vector is *"encoded"*. In the *"Maximization"* stage, the codevectors are updated with the means of the appointed vectors. This greedy algorithm converges to a local minimum. In RVQ, this iterative process is applied separately at each layer. In the proposed method however, we adapt this iterative approach of K-Means to the hierarchical layer structure of RVQ.

As in K-Means, we start with random initializations of codevectors. We use the initialization scheme proposed in K-

Means++ [23]. First we initialize the upper layers, calculate the residual for each sample and move on to the next layer, repeating this for all the layers. After the initialization, the iterations start with the first stage, where each vector is assigned to its corresponding codevector. Then the residual is calculated and transferred to the next layer. Again this process is repeated for all layers. Note that, this corresponds to "*encoding*" the sample vectors using the codebooks at hand for that iteration. Next comes the second stage, where codevectors are updated with the means of the assigned vectors, according to the assignments calculated in the first stage. This process is repeated iteratively. The pseudo-code for the iterative joint training algorithm is given in **TABLE I.**

As it can be seen, the training process of Joint K-Means is quite straightforward. It is an extension of the Expectation-Maximization algorithm to multi-layer VQ for ANN. Thanks to the proposed joint scheme, the upper layers are taken into account in the training of lower layers, which generates better codebooks and prevents overfitting. Another advantage of the proposed training algorithm is that, it includes the encoding step into the training stage. In this study, we propose also an alternative encoding method called *Joint Encoding*, which takes the lower layers into account in encoding of upper layers. Incorporating this encoding method into the training results in even better codebooks.

TABLE I. PSEUDO CODE FOR JOINT TRAINING ALGORITHM

Given: Samples, Number of Layers, Number of Codevectors
Returns: Codebooks

- Initialize Codebooks
- *For* 1:$N_{Iterations}$
  - Codes = *encode*(Samples)
  - Residuals = Samples
  - CodebookUpdate = 0
  - ClusterPopulation = 0
  - *For l* = 1:*length*(Codebooks)
    - *For r* = 1:*length*(Residuals)
      - *i* = Code[*r*][*l*]
      - CodebookUpdate[*l*][*i*] += Residuals[*r*]
      - Residuals[*r*] -= Codebook[*l*][*i*]
      - ClusterPopulation[*l*][*i*]++
    - *For k* = 1:*length*(Codebook[*l*])
      - CodebookUpdate[*l*][*k*] /= ClusterPopulation[*l*][*k*]
  - Codebooks = CodebookUpdate
- *return* Codebooks

3.2 Joint Encoding

As mentioned above, in RVQ the encoding is performed separately in each layer. However, selection of the best codevector for each layer strongly depends on the selection of previous codevectors, as the residual for a layer is obtained by subtraction of previously selected codevector from the given sample. RVQ assumes that, selecting the nearest codevector for a given residual in each layer provides a near optimal solution for the codevector selection. Here we propose an encoding scheme, which uses a heuristic search, in order to obtain a better selection.

In the proposed encoding method we assume that the codevector, which will result in the minimum quantization error, should be among a small neighborhood of the nearest codevector of the given sample. So instead of picking the nearest codevector and moving to the next layer, we select the nearest H codevectors and calculate corresponding residuals. As the residuals are transferred to the next layer, for each of them, again best H residuals are calculated. Among the H^2 options, we keep the best H with the minimum quantization error. This is repeated for all the layers. Note that this encoding scheme is similar to the beam search encoding proposed in [17], but thanks to the hierarchical structure, the search space is much more limited, hence it is computationally much cheaper. The pseudocode for the encoding scheme is given in **TABLE II.**

As explained above, the proposed Joint Encoding scheme takes into the lower layers account, resulting in lower quantization errors. Together with the proposed training scheme, final quantization error is reduced significantly. In the next section we show that the proposed performs significantly better than RVQ and the other state-of-the-art methods.

TABLE II. PSEUDO CODE FOR JOINT ENCODING ALGORITHM

Given: Sample, Codebooks
Returns: Code

- Codes = *FindNearestHCodevectors*(Sample,Codebooks[1])
- *For m* = 2:*length*(Codebooks)
  - *For h* = 1:*H*
    - *For c* = 1:*length*(Codes[*h*])
      - *i* = Codes[*h*][*c*]
      - Residuals[*h*] = Sample – Codebooks[*c*][*i*]
  - *For h* = 1:*H*
    - Codes [*h*] ← *FindNearestHCodevectors*(Residuals[*h*], Codebooks[*m*])
  - Codes = *FindBestHCodes*(Codes)
- *return* Codes[1]

4 Experiments

The proposed method is experimentally tested on two publicly available large-scale datasets: SIFT1M and GIST1M [8]. Both datasets consist of 1 Million samples. SIFT1M dataset contains 128-dimensional SIFT vectors, whereas GIST1M contains 960-dimensional GIST vectors. Each dataset has a separate training and a query set. The number of vectors in the training set for SIFT1M dataset is 100.000 and for the GIST1M dataset it is 500.000. SIFT1M consists of 10.000 queries while GIST1M includes 1000 query vectors.

4.1 Parameters

We use the given training sets to train our method and perform exhaustive search on both datasets. We use $K = 256$ codevectors for each layer, and $M = 8$ (4) layers for 64-bit (32-bit) encoding, as commonly preferred in the literature [16]–[19]. During our experiments, we observed that using higher number of residual candidates in the training stage causes only a certain number of codevectors to be updated, resulting in diverging quantization error. Hence we choose to use $H = 8$ for SIFT1M and $H = 4$ for GIST1M datasets in the training stage, as these parameters provide the lowest

quantization error. We use $H = 32$ in the encoding step for both datasets.

4.2 Retrieval Performance

The performance of the proposed method *Joint K-Means* (**JKM**) is compared against the recent state-of-the-art methods such as *Product Quantization* (**PQ**) [8], *Cartesian K-Means* (**CKM**) [14], *Transform Coding* (**TC**) [15], *Residual Vector Quantization* (**RVQ**) [16], *Additive Quantization* (**AQ/APQ**) [17], *Composite Quantization* (**CQ**) [18], *Optimized Tree Quantization* (**OTQ**) [19] and *Optimized Cartesian K-Means* (**OCK**) [20]. The results of the competing methods are taken from the figures presented in the original publications. We use the **recall@R** measure as proposed in [8]. **recall@R** can be defined as the average recall score for the true nearest neighbor in the top R retrievals. The results using 64-bit encoding for the SIFT1M and GIST1M datasets are presented in **TABLE III.** and **TABLE IV.** The results using 64-bit encoding are presented in **TABLE V.** and **TABLE VI.**

TABLE III. RESULTS ON SIFT1M FOR 64-BIT ENCODING

Method	recall@1	recall@10	recall@100
TC	0.205	0.534	0.876
RVQ	0.257	0.659	0.952
PQ	0.224	0.599	0.924
CKM	0.243	0.638	0.940
OCK	0.274	0.680	0.945
CQ	0.288	0.716	0.966
APQ	0.298	0.741	0.972
OTQ	0.317	0.748	0.972
JKM	**0.323**	**0.759**	**0.980**

TABLE IV. RESULTS ON GIST1M FOR 64-BIT ENCODING

Method	recall@1	recall@10	recall@100
TC	0.096	0.223	0.547
RVQ	0.113	0.325	0.676
PQ	0.076	0.218	0.504
CKM	0.118	0.334	0.715
OCK	0.130	0.358	0.720
CQ	0.135	0.377	0.729
AQ/APQ	N/A	N/A	N/A
OTQ	N/A	N/A	N/A
JKM	**0.140**	**0.413**	**0.769**

TABLE V. RESULTS ON SIFT1M FOR 32-BIT ENCODING

Method	recall@1	recall@10	recall@100
TC	0.057	0.197	0.519
RVQ	N/A	N/A	N/A
PQ	0.052	0.230	0.595
CKM	0.068	0.273	0.658
OCK	N/A	0.348	0.742
CQ	N/A	N/A	N/A
AQ	0.106	**0.415**	**0.825**
OTQ	0.093	0.368	0.793
JKM	**0.121**	0.402	0.790

TABLE VI. RESULTS ON GIST1M FOR 32-BIT ENCODING

Method	recall@1	recall@10	recall@100
TC	0.053	0.104	0.291
RVQ	N/A	N/A	N/A
PQ	0.023	0.068	0.176
CKM	0.054	0.142	0.396
OCK	N/A	0.172	0.467
CQ	N/A	N/A	N/A
AQ	0.069	0.189	0.467
OTQ	N/A	N/A	N/A
JKM	**0.077**	**0.213**	**0.511**

As it can be observed in the presented results, the proposed method *JKM* improves the performance of RVQ considerably. *JKM* also outperforms the state-of-the-art methods on both datasets using 64-bit encoding. *JKM* also performs better than all the compared methods for all performance measures, using 32-bit encoding on GIST1M. For 32-bit encoding on SIFT1M, *JKM* obtains the best result only for recall@1.

In order to emphasize the effectiveness of the proposed training and encoding methods, we compare the contributions of training and encoding separately, taking the performance of RVQ as a baseline. The comparison is presented in **TABLE VII.** First we compare RVQ with *JKM* using joint training only, i.e., the number of candidates for residuals $H = 1$. As it can be seen, the joint training itself without the joint encoding improves the performance of RVQ. We also test RVQ with joint encoding. The joint encoding increases the performance of RVQ significantly, yet it is not enough to outperform the state-of-the-art. Together with joint training, the state-of-the-art performance is achieved.

TABLE VII. EFFECTIVENESS OF JOINT APPROACHES, 64-BIT ENCODING, SIFT1M

Method	recall@1	recall@10	recall@100
RVQ	0.257	0.659	0.952
JKM H = 1	0.279	0.684	0.959
RVQ H = 32	0.306	0.733	0.974
JKM H = 32	**0.323**	**0.759**	**0.980**

5 Conclusion

In this paper, a novel vector quantization algorithm called Joint K-Means is proposed for approximate nearest neighbor

search on large-scale datasets. The proposed algorithm integrates the Expectation-Maximization approach to the multi-layer structure of RVQ, resulting in a joint training scheme. An alternative encoding method is also proposed where the codevector selection is performed taking the lower layers into account. The experiments conducted on publicly available datasets show that the Joint K-Means outperforms the state-of-the-art methods. In the future, we aim to investigate the scalability of this algorithm for billion scale datasets.

References

[1] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for Similarity Search: A Survey," in *arXiv preprint*, 2014, p. :1408.2927.

[2] J. He, S.-F. Chang, R. Radhakrishnan, and C. Bauer, "Compact hashing with joint optimization of search accuracy and time," in *CVPR*, 2011, pp. 753–760.

[3] J. Heo, Y. Lee, and J. He, "Spherical hashing," in *CVPR*, 2012.

[4] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognit. Lett.*, vol. 31, no. 11, pp. 1348–1358, Aug. 2010.

[5] D. Zhang, J. Wang, D. Cai, and J. Lu, "Self-Taught Hashing for Fast Similarity Search," in *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010, pp. 18–25.

[6] B. Xu, J. Bu, Y. Lin, C. Chen, X. He, and D. Cai, "Harmonious hashing," in *International Joint Conference on Artificial Intelligence*, 2013, pp. 1820–1826.

[7] G. Lin, C. Shen, D. Suter, and A. van den Hengel, "A General Two-Step Approach to Learning-Based Hashing," in *ICCV*, 2013, pp. 2552–2559.

[8] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–28, Jan. 2011.

[9] L. Zhang, Y. Zhang, J. Tang, K. Lu, and Q. Tian, "Binary code ranking with weighted hamming distance," in *CVPR*, 2013, no. 6, pp. 1586–1593.

[10] H. Jegou, M. Douze, C. Schmid, and P. Perez, "Aggregating local descriptors into a compact image representation," in *CVPR*, 2010, pp. 3304–3311.

[11] A. Gordo, F. Perronnin, Y. Gong, and S. Lazebnik, "Asymmetric distances for binary embeddings.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 1, pp. 33–47, Jan. 2014.

[12] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized Product Quantization for Approximate Nearest Neighbor Search," in *CVPR*, 2013, pp. 2946–2953.

[13] J.-P. Heo, Z. Lin, and S.-E. Yoon, "Distance Encoded Product Quantization," in *CVPR*, 2014, pp. 2139–2146.

[14] M. Norouzi and D. J. Fleet, "Cartesian K-Means," in *CVPR*, 2013, pp. 3017–3024.

[15] J. Brandt, "Transform coding for fast approximate nearest neighbor search in high dimensions," in *CVPR*, 2010, pp. 1815–1822.

[16] Y. Chen, T. Guan, and C. Wang, "Approximate nearest neighbor search by residual vector quantization," *Sensors*, vol. 10, no. 12, pp. 11259–11273, 2010.

[17] A. Babenko and V. Lempitsky, "Additive Quantization for Extreme Vector Compression," in *CVPR*, 2014, pp. 931–938.

[18] T. Zhang, D. Chao, and J. Wang, "Composite Quantization for Approximate Nearest Neighbor Search," in *ICML*, 2014.

[19] A. Babenko and V. Lempitsky, "Tree Quantization for Large-Scale Similarity Search and Classification," in *CVPR*, 2015.

[20] J. Wang, J. Wang, J. Song, X.-S. Xu, H. T. Shen, and S. Li, "Optimized Cartesian K-Means," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 180–192, Jan. 2015.

[21] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Trans. Inf. Theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[22] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognit. Lett.*, vol. 31, no. 8, pp. 651–666, 2010.

[23] D. Arthur, D. Arthur, S. Vassilvitskii, and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *SODA*, 2007, vol. 8, pp. 1027–1035.

V

COMPETITIVE QUANTIZATION FOR APPROXIMATE NEAREST NEIGHBOR SEARCH

by

E.C.Ozan, S. Kiranyaz and M. Gabbouj, 2016

IEEE Transactions on Knowledge and Data Engineering (TKDE), vol. 28, no. 11, pp. 2884 - 2894.

Competitive Quantization for Approximate Nearest Neighbor Search

Ezgi Can Ozan, Serkan Kiranyaz, *Senior*, *IEEE* and Moncef Gabbouj, *Fellow*, *IEEE*

Abstract—In this study, we propose a novel vector quantization algorithm for Approximate Nearest Neighbor (ANN) search, based on a joint competitive learning strategy and hence called as competitive quantization (CompQ). CompQ is a hierarchical algorithm, which iteratively minimizes the quantization error by jointly optimizing the codebooks in each layer, using a gradient decent approach. An extensive set of experimental results and comparative evaluations show that CompQ outperforms the-state-of-the-art while retaining a comparable computational complexity.

Index Terms— Approximate Nearest Neighbor Search, Binary Codes, Large-Scale Retrieval, Vector Quantization

THE vast increase in size and dimension of today's datasets bring about new problems, as traditional methods fail to satisfy the present-day requirements. Like many others, the problem of fast and efficient distance calculations between pairs of samples leads researchers to approximate solutions. Binary embedding of descriptor vectors for faster distance calculations has become a highly popular research topic in recent years, drawing a significant attention [1]. The common approach is to encode the vectors as binary strings and compress very large datasets in much smaller sizes, decreasing the storage cost. Furthermore, the approximation of the distance between two vectors by using pre-calculated distance values gives a significant boost in terms of the search speed.

Binary embedding methods can be divided into two major branches as hashing and vector quantization. Hashing based approaches aim to approximate the distance between vectors using the Hamming distance [2]–[6]. These approaches are proved to be fast, as the Hamming distance calculation between two binary strings is basically an XOR operation, but since the Hamming distance is an integer between 0 and the length of the binary string, several vector pairs end up with the exact same distance approximation, which is a disadvantage in terms of retrieval rankings. Approaches such as [7]–[9] apply weighted Hamming distances, in order to add more variety to the results of the distance approximation by using look-up tables, which improves the performance; however, this slows down the search since distance approximations cannot be calculated simply by an XOR operation.

The introduction of weighted distances and look-up tables for hashing opened the doors for new binary embedding approaches, which constitutes the second major branch: the Vector Quantization (VQ). VQ is a very well-studied area in many fields such as electronics, telecommunication and signal processing. The application of VQ in binary embedding methods for approximate distance calculations, or as more formally stated in the literature, Approximate Nearest Neighbor (ANN) search starts with the Product Quantization (PQ) proposed in [10]. In this study, Jégou *et al.* propose a division among the dimensions of the vector. They perform quantization separately at each subspace, and obtain the final quantized vector as the Cartesian products of the sub-quantized vectors. This opens a new era in quantization for ANN as with this method, the number of quantization centers can scale up to very large numbers. Several improvements have been applied on top of PQ such as [11]–[15], obtaining significant increase in performance.

Besides PQ and its variants, for the purpose of scaling up the number of quantization codevectors exponentially, another approach is to quantize a given vector as the *addition* of several codevectors. Chen *et al.* in [16] propose this approach to ANN search problems. The authors introduce several layers of quantization, as each layer quantizes the residuals of the previous layer. Improvements on this method have been proposed in [17], [18]. In [19], Babenko *et al.* propose a method again based on quantization by addition of several codevectors, yet they remove the hierarchy between layers of residual quantization, providing a high level of freedom for the quantization codevectors. However, this freedom requires an encoding step, which is computationally very expensive, but still it outperforms the state-of-the-art methods. Other methods such as [20]–[22] add further constraints on the selection of codevectors, in order to obtain a more efficient encoding step.

In summary, many recent methods propose quantization through the addition of several codevectors, but they have to choose between using either highly constrained codevector proposals as in [16], which results in inferior performance; or leaving more freedom for codevectors while looking for a heuristic search, which results in computationally expensive encoding. To address this drawback in an efficient way we propose a novel VQ approach, which outperforms the state-

- *Ezgi Can Ozan, is with the Department of Signal Processing, Tampere University of Technology, Tampere, Finland (e-mail: ezgi.ozan@tut.fi).*
- *Prof. Serkan Kiranyaz, is with Electrical Engineering Department, College of Engineering, Qatar University, Qatar. (e-mail: mkiranyaz@qu.edu.qa).*
- *Prof. Moncef Gabbouj, is with the Department of Signal Processing, Tampere University of Technology, Tampere, Finland (e-mail: moncef.gabbouj@tut.fi).*

of-the-art quantization methods, with a comparable computational complexity. The novel contributions in this paper can be summarized as follows:

- Proposing a novel addition based quantization method, which preserves the layer hierarchy as in [16], consisting of a novel method which jointly trains all the codebooks, and minimizes the quantization error together in all layers.
- Proposing a better encoding approach by redefining *'the winner codevector'*, which provides lower quantization error.

The rest of the paper is organized as follows. In **Section 1**, the problem formulation is given together with a more detailed explanation of the related work. In **Section 2**, the proposed method is presented in detail. **Section 3** presents the experimental results and comparisons with the state-of-the-art. In **Section 0**, the method and the obtained results are discussed thoroughly, and finally **Section 5** concludes the paper.

1 Problem Formulation and Related Work

Vector quantization (VQ) can be seen as an optimization problem, where the optimization criterion is to minimize the mean squared quantization error. The quantization error of a quantizer Q can be described as follows: Given a set of N vectors $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$, the mean squared quantization error MSE_Q is defined as in (1).

$$MSE_Q = \frac{1}{N}\sum_{i}^{N} \|\boldsymbol{x}_i - Q(\boldsymbol{x}_i)\|_2^2 \tag{1}$$

The quantizer Q quantizes the given feature vector $\boldsymbol{x}_i$ to its corresponding codevector as,

$$Q(\boldsymbol{x}_i) = \boldsymbol{C}\boldsymbol{b}_i \tag{2}$$

where $\boldsymbol{b}_i \in \{0,1\}^K$ is the binary selection vector, with $\|\boldsymbol{b}_i\|_1 = 1$. The main difference between traditional VQ and VQ for ANN is that, the number of codevectors for ANN is much greater than traditional vector quantization, because VQ for ANN targets large-scale datasets. As mentioned earlier, in order to increase the number of codevectors exponentially, VQ methods for ANN usually use several codebooks for quantization [1]. The use of M codebooks increase the number of codevectors from K to K^M. Codevectors from different codebooks are combined either by concatenation (Cartesian products) as in [10], [12]–[14], or by addition as in [16], [19]–[22].

Vector quantization by Cartesian product of codevectors can be formulated as,

$$\hat{\boldsymbol{x}}_i = \begin{bmatrix} \boldsymbol{C}^{(1)}\boldsymbol{b}_i^{(1)} \\ \vdots \\ \boldsymbol{C}^{(M)}\boldsymbol{b}_i^{(M)} \end{bmatrix} \qquad \boldsymbol{C}^{(m)}\boldsymbol{b}_i^{(m)} \in \mathbb{R}^{D/M} \tag{3}$$

where vector, $\hat{\boldsymbol{x}}$, is the quantization output of vector $\boldsymbol{x}$. $\boldsymbol{C}^{(m)} = \left\{\boldsymbol{c}_1^{(m)} \dots \boldsymbol{c}^{(m)}{}_K\right\} \in \mathbb{R}^{D/M \times K}$ is the corresponding codebook of the m^{th} subspace, $m \in \{1 \dots M\}$, where M is the number of codebooks (one codebook per subspace) and K is the number of codevectors per codebook. $\boldsymbol{B}^{(m)} = \left\{\boldsymbol{b}_1^{(m)} \dots \boldsymbol{b}_N^{(m)}\right\} \in \mathbb{R}^{K \times N}$ is the set of K-dimensional, binary codevector selection vectors for the m^{th} codebook, where $\boldsymbol{b}_i^{(m)} \in \{0,1\}^K$ with $\left\|\boldsymbol{b}_i^{(m)}\right\|_1 = 1$.

Similarly, vector quantization by addition of codevectors can be formulated as given in (4). The main difference from (3) is that here all codebooks come from the original feature space, i.e., $\boldsymbol{C}_m = \{\boldsymbol{c}_{m_1} \dots \boldsymbol{c}_{m_K}\} \in \mathbb{R}^{D \times K}$. Note that in this paper, codebooks obtained from the same feature space are represented using a subscript ($\boldsymbol{C}_m$), while codebooks belonging to different subspaces are represented using a superscript in parenthesis ($\boldsymbol{C}^{(m)}$). The corresponding binary selection vectors are also represented accordingly.

$$\hat{\boldsymbol{x}}_i = \sum_{m=1}^{M} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \qquad \boldsymbol{C}_m \boldsymbol{b}_{m_i} \in \mathbb{R}^D \tag{4}$$

Different codevectors obtained from different codebooks are combined in order to obtain $\hat{\boldsymbol{x}}$, the final approximation of vector $\boldsymbol{x}$, providing the minimum quantization error. This optimization problem can be formulated for the Cartesian product based VQ as given in (5), and for the addition based VQ as given in (6).

$$MSE_Q^{(Cartesian)} = \min_{\{\boldsymbol{C}^{(m)}\},\{\boldsymbol{B}^{(m)}\}} \frac{1}{N}\sum_{i=1}^{N} \left\| \boldsymbol{x}_i - \begin{bmatrix} \boldsymbol{C}^{(1)}\boldsymbol{b}_i^{(1)} \\ \vdots \\ \boldsymbol{C}^{(M)}\boldsymbol{b}_i^{(M)} \end{bmatrix} \right\|_2^2 \tag{5}$$

$$MSE_Q^{(Addition)} = \min_{\{\boldsymbol{C}_m\},\{\boldsymbol{B}_m\}} \frac{1}{N}\sum_{i=1}^{N} \left\| \boldsymbol{x}_i - \sum_{m=1}^{M} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \right\|_2^2 \tag{6}$$

Closed form solutions for the above problems do not exist, hence the recent methods approach this minimization using approximate techniques with different constraints and different heuristics. Constraints limit the search space and heuristics enable a more feasible and divergent search. In the next subsection, the state-of-the-art methods will be examined in terms of such constraints and the heuristics they define.

1.1 Product Quantization

Product Quantization (PQ) [10] proposed by Jegou *et al.* is a Cartesian product based VQ method. It uses subspaces of the feature space to create different layers of quantization. On each subspace, a different codebook is trained separately. In other words, PQ divides the original feature space into M subspaces, and each codevector $\boldsymbol{c}_k^{(m)} \in \mathbb{R}^{D/M}$ obtained on this subspace is a subvector. This approach splits the optimization problem into M independent problems. However, the assumption of subspaces being statistically independent does not usually hold in practice, and many variants of PQ have been proposed [11]–[14] in order to overcome this drawback. The optimization problem of PQ can be formulated as given in (7), where $\boldsymbol{x}_i^{(m)}$ is the subvector of $\boldsymbol{x}_i$ corresponding to the m^{th}

subspace.

$$\begin{aligned} MSE_Q^{(PQ_1)} &= \min_{\{\boldsymbol{C}^{(1)}\},\{\boldsymbol{B}^{(1)}\}} \frac{1}{N}\sum_{i=1}^{N} \left\| \boldsymbol{x}_i^{(1)} - \boldsymbol{C}^{(1)}\boldsymbol{b}_i^{(1)} \right\|_2^2 \\ &\cdot \\ &\cdot \\ MSE_Q^{(PQ_M)} &= \min_{\{\boldsymbol{C}^{(M)}\},\{\boldsymbol{B}^{(M)}\}} \frac{1}{N}\sum_{i=1}^{N} \left\| \boldsymbol{x}_i^{(M)} - \boldsymbol{C}^{(M)}\boldsymbol{b}_i^{(M)} \right\|_2^2 \end{aligned} \tag{7}$$

1.2 K-Subspace Quantization

In Cartesian product based approaches, the creation of subspaces is problematic because of the unrealistic assumption that subspaces are statistically independent. The proposed solution to this problem is a PCA transformation [11]–[14], which brings another problem of unbalanced distribution of information among dimensions. In [15], Ozan *et al.* also target this problem and propose to introduce more than one affine subspace, train a Transform Coding [11] variant in each and choose the best among them to quantize the samples. Introduction of multiple affine subspaces improves the assumption of independent subspaces after PCA. Besides, they propose to update codebooks in an iterative way, in order to minimize the quantization error further and obtain the state-of-the-art performance. The optimization problem for KSSQ can be formulated as given in (8). $\boldsymbol{X}_m \subset \boldsymbol{X}$ is a subset of samples such that $\bigcup_{m=1}^{M} \boldsymbol{X}_m = \boldsymbol{X}$ and $\boldsymbol{X}_m \cap \boldsymbol{X}_j = \emptyset$ where $m \neq j$. L_m is the number of dimensions for the m^{th} subspace.

$$\begin{aligned} MSE_Q^{(KSSQ)} = \min_{\left\{\boldsymbol{C}_m^{(\mathrm{l}_m)}\right\},\left\{\boldsymbol{B}_m^{(\mathrm{l}_m)}\right\},\{\boldsymbol{X}_m\},\{\mathrm{L}_m\}} \frac{1}{N}\sum_{m=1}^{M}\sum_{\boldsymbol{x}_i \in \boldsymbol{X}_m} \Bigg\| \boldsymbol{x}_i \\ - \begin{bmatrix} \boldsymbol{C}_m^{(1)}\boldsymbol{b}_{m_i}^{(1)} \\ \cdot \\ \cdot \\ \boldsymbol{C}_m^{(\mathrm{L}_m)}\boldsymbol{b}_{m_i}^{(\mathrm{L}_m)} \end{bmatrix} \Bigg\|_2^2 \end{aligned} \tag{8}$$

1.3 Residual Vector Quantization

Residual Vector Quantization (RVQ) [16] proposed by Chen *et al.* is an addition based VQ method, which dictates a hierarchy among the codebooks. As the name suggests, each succeeding codebook is trained on the residuals of the previous layer. The optimization problem is separated into M subproblems, which are solved consecutively. The optimization problem of RVQ can be formulated as,

$$\begin{aligned} MSE_Q^{(RVQ_1)} &= \min_{\{\boldsymbol{C}_1\},\{\boldsymbol{B}_1\}} \frac{1}{N}\sum_{i=1}^{N} \left\| \left(\boldsymbol{x}_i - \sum_{m=1}^{1} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \right) \right\|_2^2 \\ &\cdot \\ &\cdot \\ MSE_Q^{(RVQ_M)} &= \min_{\{\boldsymbol{C}_M\},\{\boldsymbol{B}_M\}} \frac{1}{N}\sum_{i=1}^{N} \left\| \left(\boldsymbol{x}_i - \sum_{m=1}^{M} \boldsymbol{C}_m \boldsymbol{b}_{m_i} \right) \right\|_2^2 \end{aligned} \tag{9}$$

This hierarchy simplifies the encoding process, as each codevector depends on the selection of the previous codevectors. However, the contribution of each layer to the minimization of the quantization error is not the same, as the last layers contribute much less than the first ones. This algorithm has been extended by several other methods [17], [18].

1.4 Optimized Cartesian K-Means

Optimized Cartesian K-Means (OCKM) [20] proposed by Wang *et al.* is an improvement over the Cartesian K-Means (CKM) [13], which forms the quantization vector as a Cartesian product of subvectors obtained from subspaces of the original feature space. Similar to CKM, OCKM proposes a multiplication with a rotation matrix $\boldsymbol{R}$ before dividing the initial vector space into subspaces. OCKM is a transition between the Cartesian product based vector quantization algorithms and the addition based vector quantization algorithms. Wang *et al.* propose to apply the same rotation and subspace division in CKM, but instead of picking one codevector per subspace, the authors suggest training two codebooks per subspace, hence picking two codevectors and adding them up. The optimization performed by OCKM can be formulated by (10).

$$MSE_Q^{(OCKM)} = \min_{\left\{\boldsymbol{C}_c^{(m)}\right\},\left\{\boldsymbol{B}_c^{(m)}\right\},\{\boldsymbol{R}\}} \frac{1}{N}\sum_{i=1}^{N} \left\| \boldsymbol{x}_i - \boldsymbol{R} \begin{bmatrix} \sum_{c=1}^{2} \boldsymbol{C}_c^{(1)}\boldsymbol{b}_{c_i}^{(1)} \\ \cdot \\ \sum_{c=1}^{2} \boldsymbol{C}_c^{(M/2)}\boldsymbol{b}_{c_i}^{(\mathrm{M}/2)} \end{bmatrix} \right\|_2^2 \tag{10}$$

OCKM generates the codebooks in different subspaces, yet within each subspace, there is no constraint on the codebooks and the codevectors they contain. In order to avoid the complexity of this unconstrained situation, they opt for more subspaces rather than codebooks i.e., they keep the number of codebooks per subspace limited to 2.

1.5 Additive Quantization

Additive Quantization (AQ) [19] is an unconstrained approach to addition based VQ. Babenko *et al.* propose to generate the quantized vectors using the addition of several different codevectors, each from a different codebook, trained without any constraints on the original feature space. The lack of constraints provides a boost in the quantization performance, yet the training and more importantly encoding procedure is extremely costly, as they propose to use "*heuristic beam search"* in the encoding process. Since there are no specific constraints, the optimization problem can be represented as in (6).

The complexity of the proposed beam search in [19] is proportional to the cubic order of the number of codebooks, the authors propose a PQ variant of the additive quantization, Additive Product Quantization (APQ) [19], which divides the feature space into two subspaces and performs AQ in each subspace independently, then concatenates both vectors to obtain the final quantized vector. This splits the optimization problem into two independent problems similar to PQ. The optimization problem for APQ can be formulated as given in (11).

$$MSE_Q^{(APQ_1)} = \min_{\{\boldsymbol{C}_m^{(1)}\},\{\boldsymbol{B}_m^{(1)}\}} \frac{1}{N}\sum_{i=1}^{N}\left\|\boldsymbol{x}^{(1)}{}_i - \sum_{m=1}^{M/2}\boldsymbol{C}_m^{(1)}\boldsymbol{b}_{m_i}^{(1)}\right\|_2^2$$
$$MSE_Q^{(APQ_2)} = \min_{\{\boldsymbol{C}_m^{(2)}\},\{\boldsymbol{B}_m^{(2)}\}} \frac{1}{N}\sum_{i=1}^{N}\left\|\boldsymbol{x}^{(2)}{}_i - \sum_{m=1}^{M/2}\boldsymbol{C}_m^{(2)}\boldsymbol{b}_{m_i}^{(2)}\right\|_2^2 \quad (11)$$

1.6 Composite Quantization

Composite Quantization (CQ) [21] proposed by Zhang *et al.* is yet another addition based vector quantization method with constraints. CQ differs from the previous methods by the purpose of the selected constraints. Compared to the Cartesian product based methods, one serious drawback of the addition based VQ techniques is the computational cost of asymmetric distance calculation. Usually, Cartesian product based methods can calculate the approximate distance between an encoded database element and a given query vector in M look-ups and additions, i.e., one look-up and addition for each codebook [10], [12], [13]. However, the computational cost for addition based methods has usually the complexity of $O(M^2)$ [16], [17], [19]. Zhang *et al.* propose bringing additional constraints in the codebook generation process so that the sum of the dot products of all codevectors from two different codebooks is equal to a constant value, i.e., $\sum_{m=1}^{M}\sum_{l=1,l\neq m}^{M}\left(\boldsymbol{C}_m\boldsymbol{b}_{m_i}\right)^T\boldsymbol{C}_l\boldsymbol{b}_{l_i} = \epsilon$. They include this constraint into the optimization via a penalty factor. The formulation of the optimization problem can be given as in (12), where μ is the penalty parameter.

$$MSE_Q^{(CQ)} = \min_{\{\boldsymbol{C}_m\},\{\boldsymbol{B}_m\},\{\epsilon\}} \frac{1}{N}\sum_{i=1}^{N}\left\|\boldsymbol{x}_i - \sum_{m=1}^{M}\boldsymbol{C}_m\boldsymbol{b}_{m_i}\right\|_2^2 + \mu\sum_{i=1}^{N}\sum_{m=1}^{M}\sum_{l=1,l\neq m}^{M}\left(\left(\boldsymbol{C}_m\boldsymbol{b}_{m_i}\right)^T\boldsymbol{C}_l\boldsymbol{b}_{l_i} - \epsilon\right)^2 \quad (12)$$

1.7 (Optimized) Tree Quantization

Babenko *et al.* extended their previous work AQ, by adding more constraints on the codebooks and a rotation as in OPQ for the feature space, resulting with (Optimized) Tree Quantization (OTQ) [22]. In OTQ, a tree structure is introduced where each vertex of the tree is a codebook. Each dimension of the feature space is assigned to an edge in the tree, so each dimension is coded by two codebooks. It is assumed that any dimension that is not in the edge of a codebook is equal to zero. This brings orthogonality between the codevectors of any two codebooks that are not adjacent in the tree. The introduced tree structure significantly decreased the encoding complexity, while obtaining a comparable quantization performance with AQ. The optimization problem for OTQ can be formulated by (13). Here $[d]$ represents the d^{th} dimension of the feature space and $a(m,n) = 1$ if dimension d is assigned to edge (m,n).

$$MSE_Q^{(OTQ)} = \min_{\{\boldsymbol{C}_m\},\{\boldsymbol{B}_m\},\{a(m,n)\}} \sum_{i=1}^{N}\sum_{d=1}^{D}\sum_{m=1}^{M}\sum_{n=m}^{M} a(m,n) \times \left|\boldsymbol{C}_m\boldsymbol{b}_{m_i}[d] + \boldsymbol{C}_n\boldsymbol{b}_{n_i}[d] - \boldsymbol{x}_i[d]\right|^2 \quad (13)$$

A noteworthy observation from the aforementioned prior studies in this domain is that the methods with less constraints offer a better quantization performance, whereas the complexity of the encoding and distance calculation increases. The methods with stronger constraints are faster yet perform worse. In this study, we aim to achieve a superior performance without increasing the computational complexity. In order to accomplish this we start from a well-constrained quantization with a hierarchical formation and then we shall relax these constraints for a better quantization performance whilst having a comparable computational complexity.

2 The Proposed Method: Competitive Quantization

As we discussed in the previous section, constraint selection is crucial as it significantly effects the quantization performance, computational complexity and search speed. In this paper, we focus on the hierarchical structure imposed by RVQ. The biggest advantage of RVQ's hierarchical approach is that, the codebooks have a given order, so encoding is simply selecting the nearest codevector in the current layer, calculating the residual and proceeding to the next layer. However, each codebook is trained separately, independent from the others, resulting in a suboptimal solution. In other words, the codebook training in the upper layers does not take the quantization error produced by the lower layers into account. In this study, we propose a joint optimization scheme updating all layers at the same time.

2.1 Competitive Codebook Learning

For addition based hierarchically connected quantization methods, for a given vector $\boldsymbol{x}$, a residual vector $\boldsymbol{r}_m$ at the m^{th} level can be represented as given in (14).

$$\boldsymbol{r}_m = \boldsymbol{x} - \sum_{l=1}^{m-1}\boldsymbol{C}_l\boldsymbol{b}_l \quad (14)$$

"*Competitive Learning*" with a *winner-takes-all* strategy is one way to obtain the codebooks for such a connectionist quantization scheme [23]. In this scheme, each layer responds to its corresponding input by determining a winner codevector. The winner codevectors are updated and moved towards the input to minimize the error. As (14) shows, all the codevectors from all the levels are responsible from the obtained quantization error, meaning that all the codevectors should be updated together accordingly, to minimize this error.

In this paper, the codebooks are jointly optimized using the stochastic gradient decent, following the aforementioned competitive learning approach. With the stochastic gradient decent method, the codevectors are updated using the formula in (15).

$$\dot{\boldsymbol{c}}_m(t+1) = \dot{\boldsymbol{c}}_m(t) - \gamma_m(t)\nabla_{\dot{\boldsymbol{c}}_m}\left(\left\|\boldsymbol{x} - \sum_{l=1}^{M}\dot{\boldsymbol{c}}_l\right\|_2^2\right) \quad (15)$$

Here, the parameter $\gamma_m(t)$ is the learning rate at iteration t, which is decreased with every iteration. $\dot{\boldsymbol{c}}_m$ stands for the winner codevector for the sample vector $\boldsymbol{x}$ at the m^{th} layer, i.e., $\dot{\boldsymbol{c}}_m = \boldsymbol{C}_m \boldsymbol{b}_m$ or equivalently;

$$\dot{\boldsymbol{c}}_m = \underset{\boldsymbol{c}_{m_k}}{\operatorname{argmin}} \left\| \left(\boldsymbol{x} - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l \right) - \boldsymbol{c}_{m_k} \right\|_2^2 \tag{16}$$

According to (16) the gradient of the error for the sample vector $\boldsymbol{x}$ can be calculated as,

$$\nabla_{\dot{\boldsymbol{c}}_m} \left(\left\| \boldsymbol{x} - \sum_{l=1}^{M} \dot{\boldsymbol{c}}_l \right\|_2^2 \right) = 2 \left(\sum_{l=1}^{M} \dot{\boldsymbol{c}}_l - \boldsymbol{x} \right) \tag{17}$$

So, for each winner codevector at each level, the update rule, which is going to move the corresponding codevector towards the input can be formulated as,

$$\dot{\boldsymbol{c}}_m(t+1) = \dot{\boldsymbol{c}}_m(t) + 2\gamma_m(t) \left(\boldsymbol{x} - \sum_{l=1}^{M} \dot{\boldsymbol{c}}_l \right) \tag{18}$$

As the equation in (18) shows, at each iteration and for each sample, the winner codevectors in each layer should be updated by multiplying the learning rate with the error vector in order to minimize the total quantization error, as the result of the stochastic gradient descent approach. Note that, unlike RVQ, the codevectors from all layers are updated jointly. This prevents overfitting in upper levels and increases the contribution of lower levels to the minimization of the quantization error. The training algorithm including the iterative codevector updates is presented in **TABLE 1**.

TABLE 1
COMPQ TRAINING ALGORITHM

INPUT: Training set $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$
OUTPUT: Quantization codebooks $\boldsymbol{C}_m$

- Initialize the codebooks.
- *for* $\boldsymbol{N_{it}}$ iterations
  - *for* each sample $\boldsymbol{x_i}$
    - Encode $\boldsymbol{x_i}$ in order to obtain the winner codevectors as given in **Section 2.3**
    - Calculate the quantization error vector $\boldsymbol{x_i} - \sum_{m=1}^{M} \dot{\boldsymbol{c}}_m$
    - Update the winner codevectors with the error vector according to the equation in (18).
  - Decrease the learning rate by 1%.
- Return the quantization codebooks.

2.2 Initialization of Codebooks

CompQ starts with a broad initialization of the codebooks. We use Transform Coding [11] in order to generate the initial codebooks for each layer. Transform Coding (TC) is also a VQ method for ANN. Using PCA, TC transforms the feature space into a new one, in which the dimensions are orthogonal. Centroids are then determined in the transform domain for each dimension. Combining these centroids yields the codevectors. The number of centroids for each dimension is proportional to the variance of the corresponding dimension, providing a balanced distribution of codevectors in the feature space. Initially, coarsely computed codebooks are selected in order to prevent early overfitting. We follow the hierarchy as formulated in (9). We train the TC for a layer, and obtain the initial codevectors, then calculate the residual vectors and proceed to the next layer.

2.3 Sample Encoding and Winner Codevector

Encoding of a new sample consists of finding out the winner codevector for each layer. The winner codevector $\dot{\boldsymbol{c}}_m$ can be defined as,

$$\dot{\boldsymbol{c}}_m = \underset{\boldsymbol{c}_{m_k}}{\operatorname{argmin}} \left\| \left(\boldsymbol{x} - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l \right) - \boldsymbol{c}_{m_k} \right\|_2^2 \tag{19}$$

Recall that in hierarchical structure based approaches such as RVQ and its variants, for each layer, the nearest codevector on the corresponding layer is chosen as the winner codevector. However, this may not be the best choice as it may lead to higher quantization errors. A toy example for the aforementioned problem is presented in **Fig. 1**. In this figure, the given sample (X) is encoded with (A3), since it is closer to the (A) in the upper layer than (B). However, (B1) would have been a better choice, as it is the nearest code to the given sample.

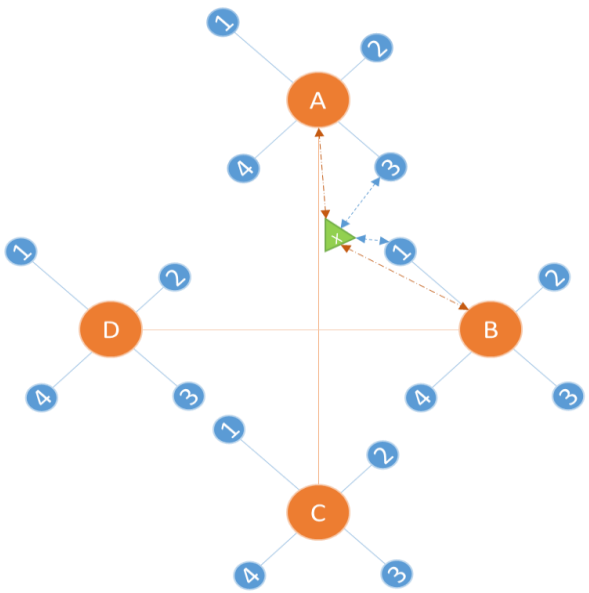

Fig. 1: Illustration of inferior encoding using hierarchical layers. The sample (X) is encoded with (A3) instead of the nearest code (B1).

CompQ follows an alternative approach for the determination of the winner codevector. For each layer, instead of one codevector, the best H candidates are picked and then the residual for each candidate is calculated. Then among HK residuals, again the best H are stored for the next layer. When the final layer is reached the winner codevectors for each layer have been obtained. Going back to **Fig. 1**, if (A) and (B)

were considered as candidate codevectors for the first layer, then in the second layer (B1) would eventually be selected, as it gives the minimum quantization error. Note that, this is a special case of the beam search algorithm used in the encoding step of AQ [19]. A very similar approach is also proposed in OCKM [20]. In this paper, the hierarchical structure is imposed on the beam search and it limits the search space, hence reduces the computational complexity drastically. Since AQ is not inclusive of any structure, the beam search in AQ searches for the best H among $(M-m+1)HK$ codevectors for the m^{th} layer. Thanks to the hierarchy in our case there are always HK vectors to compare.

The distance calculation between a codevector and a residual is formulated in (20). After some mathematical manipulations, this equation can be rewritten to enable the use of look-up tables, and accelerate the encoding, as given in (21) and (22).

$$d(\boldsymbol{r}_{m_i}, \boldsymbol{c}_{m,k}) = \left(\left\| \left(\boldsymbol{x}_i - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l - \boldsymbol{c}_{m,k} \right) \right\|_2^2 \right) \tag{20}$$

$$d(\boldsymbol{r}_{m_i}, \boldsymbol{c}_{m,k}) = \left(\left\| \boldsymbol{x}_i - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l \right\|_2^2 - 2 \left\langle \boldsymbol{x}_i - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l , \boldsymbol{c}_{m,k} \right\rangle + \left\| \boldsymbol{c}_{m,k} \right\|_2^2 \right) \tag{21}$$

$$d(\boldsymbol{r}_{m_i}, \boldsymbol{c}_{m,k}) = \left(\left\| \boldsymbol{x}_i - \sum_{l=1}^{m-1} \dot{\boldsymbol{c}}_l \right\|_2^2 - 2\langle \boldsymbol{x}_i, \boldsymbol{c}_{m,k} \rangle + 2 \sum_{l=1}^{m-1} \langle \dot{\boldsymbol{c}}_l, \boldsymbol{c}_{m,k} \rangle + \left\| \boldsymbol{c}_{m,k} \right\|_2^2 \right) \tag{22}$$

Note that in (22), the first term is already calculated in the previous layer for all H candidates. The third and fourth term can be obtained from a look-up table. Only the second term should be calculated for each encoding operation.

2.4 Asymmetric Distance Calculation

The Asymmetric Distance [9], [24] is calculated between an encoded database element and a given query vector. For simplicity we omit the square root and represent the formulation of the square of the asymmetric distance, as given below:

$$d(\boldsymbol{x}, \bar{\boldsymbol{x}})^2 = \left\| \boldsymbol{x} - \bar{\boldsymbol{x}} \right\|_2^2 = \left\| \boldsymbol{x} - \sum_{m=1}^{M} \dot{\boldsymbol{c}}_m \right\|_2^2 \tag{23}$$

$$d(\boldsymbol{x}, \bar{\boldsymbol{x}})^2 = \left\| \boldsymbol{x} \right\|_2^2 - 2 \left\langle \boldsymbol{x}, \sum_{m=1}^{M} \dot{\boldsymbol{c}}_m \right\rangle + \left\| \sum_{m=1}^{M} \dot{\boldsymbol{c}}_m \right\|_2^2 \tag{24}$$

$$d(\boldsymbol{x}, \bar{\boldsymbol{x}})^2 = \left\| \boldsymbol{x} \right\|_2^2 - 2 \sum_{m=1}^{M} \langle \boldsymbol{x}, \dot{\boldsymbol{c}}_m \rangle + \sum_{m=1}^{M} \sum_{l=1}^{M} \langle \dot{\boldsymbol{c}}_m, \dot{\boldsymbol{c}}_l \rangle \tag{25}$$

As shown in (25), two look-up tables must be prepared for distance calculation beforehand. The first look-up table consists of dot-products of the query vector $\boldsymbol{x}$ with all codevectors. The second look-up table stores the dot-products calculated between pairs of codevectors. Since the first term of (25) is the same for all database elements, it can be neglected. The distance can be then calculated by $M + M^2$ look-ups and additions as in [16]–[19].

3 Experimental Results

3.1 Exhaustive Search

Our approach is first tested on two publicly available datasets of 1 Million samples, SIFT1M and GIST1M [10] for exhaustive search. SIFT1M consists of 128-dimensional SIFT vectors and GIST1M consists of 960-dimensional GIST vectors.

We train our method using the given training sets and perform exhaustive search on both datasets for all given queries. We use $K = 256$, $H = 32$, $M = 8$ for 64-bits and $M = 4$ for 32-bits coding. The performance of CompQ is compared against the recent state-of-the-art methods such as, *Residual Vector Quantization* (**RVQ**) [16], *Cartesian K-Means/Optimized Product Quantization* (**CKM/OPQ**) [12], [13], *Additive Quantization* (**AQ/APQ**) [19], *Composite Quantization* (**CQ**) [21], *Extended Residual Vector Quantization* (**ERVQ**) [17], *Optimized Cartesian K-Means* (**OCK**) [20] and *Optimized Tree Quantization* (**OTQ**) [22]. We do not compare against *Transform Coding* [11], *Product Quantization* [10], *Projected Residual Vector Quantization* [18] and *Distance Encoded Product Quantization* [25] since their results were already outperformed by the other compared methods. The results of the competing methods are taken from the original publications. For AQ/APQ, AQ is used for 32-bits coding and APQ for 64-bits as suggested by the authors. *NA* corresponds to missing results in the original publications.

The **recall@*R*** measure is used for the experiments, which is the recall value for the first R samples in retrieval. The nearest sample in the test set is taken as the ground truth for each query. We present the results for **recall@1, recall@10** and **recall@100** for 32-bit coding in **TABLE 2** and **TABLE 3** and for 64-bit coding in **TABLE 4** and **TABLE 5**, respectively.

As observed from the results, the proposed method, *CompQ*, outperforms all recent state-of-the-art methods for all scores, on both datasets for 64-bits encoding. For 32-bits encoding, *CompQ* still has the best performance for recall@10 for SIFT1M and also for recall@10 and recall@100 for GIST1M. *KSSQ* gives slightly better results than *CompQ* for recall@1 in both datasets for 32-bits encoding.

TABLE 2
TEST RESULTS FOR SIFT1M, 32-BIT CODES

	recall@1	recall@10	recall@100
RVQ	*NA*	*NA*	*NA*
CKM/OPQ	0.068	0.273	0.658
AQ	0.106	0.415	**0.825**
CQ	*NA*	*NA*	*NA*
OCK	*NA*	0.348	0.742
ERVQ	*NA*	*NA*	*NA*
OTQ	0.093	0.368	0.793
KSSQ	**0.145**	0.434	0.802
CompQ	0.135	**0.435**	0.818

TABLE 3
TEST RESULTS FOR GIST1M, 32-BIT CODES

	recall@1	recall@10	recall@100
RVQ	*NA*	*NA*	*NA*
CKM/OPQ	0.054	0.142	0.396
AQ	0.069	0.189	0.467
CQ	*NA*	*NA*	*NA*
OCK	*NA*	0.172	0.467
ERVQ	*NA*	*NA*	*NA*
OTQ	*NA*	*NA*	*NA*
KSSQ	**0.078**	0.191	0.437
CompQ	0.072	**0.200**	**0.504**

TABLE 4
TEST RESULTS FOR SIFT1M, 64-BIT CODES

	recall@1	recall@10	recall@100
RVQ	0.257	0.659	0.952
CKM/OPQ	0.243	0.638	0.940
APQ	0.298	0.741	0.972
CQ	0.288	0.716	0.967
OCK	0.274	0.680	0.945
ERVQ	0.276	0.694	0.962
OTQ	0.317	0.748	0.972
KSSQ	0.325	0.754	0.976
CompQ	**0.352**	**0.795**	**0.987**

TABLE 5
TEST RESULTS FOR GIST1M, 64-BIT CODES

	recall@1	recall@10	recall@100
RVQ	0.113	0.325	0.676
CKM/OPQ	0.118	0.334	0.715
AQ/APQ	*NA*	*NA*	*NA*
CQ	0.135	0.377	0.729
OCK	0.130	0.358	0.720
ERVQ	0.115	0.341	0.711
OTQ	*NA*	*NA*	*NA*
KSSQ	0.136	0.396	0.741
CompQ	**0.155**	**0.419**	**0.801**

3.2 Non-Exhaustive Search

As mentioned above in **Section 2.4**, the asymmetric distance calculation requires $M + M^2$ look-ups and additions as in many addition based methods such as [16]–[19]. This means the exhaustive search using those methods takes longer time compared to the Cartesian product based methods, which generally requires M look-ups and additions. Methods such as CQ and OTQ propose additional constraints on codebooks to decrease this cost. In our method, we propose non-exhaustive search as a solution to this problem.

Non-exhaustive versions of methods such as PQ, OPQ have been implemented and tested using an additional coarse quantization layer as in [10], [12], [14], [26], [27]. However, in our method, a non-exhaustive search scheme can be implemented using the hierarchical structure. Since at each layer, residuals of the previous layers are quantized, upper layers can be used as coarse quantization layers and an inverted file list can be created, i.e., no additional coarse quantizer required.

Here we should state that, other non-exhaustive search algorithms such as [14], [26]–[29] can also be applied on top of the proposed method, where the proposed method can be used as a subquantizer or for re-ranking purposes. Here we only discuss the non-exhaustive implementation of the proposed method, as this is an inherent property of it. We do not aim to propose a new indexing algorithm, as it would be beyond the scope of this paper.

A non-exhaustive version of RVQ (IVFRVQ) has been also proposed in [16], which uses the first L layers as inverted file indexes. The query vector is compared to K^L codevectors and the nearest W codevectors are selected ($W < K$). In our method, we propose a minor modification to decrease this initial comparison overhead. The query vector is compared to the codevectors of the initial layer and the best W of them are selected, and residuals for the second layer are calculated. Then among KW residuals, the nearest W^2 are selected as target inverted file indexes, while in [16], the query is compared to all codevectors in the first two layers, resulting in K^2 comparisons. The comparison of the proposed non-exhaustive search algorithm with different W values is given in TABLE **6**.

TABLE 6
NON-EXHAUSTIVE SEARCH, SIFT1M, $L = 2$, 64-BIT CODES

	Avg. No. Comparisons	Avg. Speed-Up	recall		
			@1	@10	@100
W=8	4115	**x 146**	0.305	0.622	0.707
W=16	12788	x 65	0.343	0.742	0.886
W=32	37951	x 25	0.351	0.786	0.964
W=64	108064	x 9	**0.352**	**0.795**	0.986
Exhaustive	1000000	x 1	**0.352**	**0.795**	**0.988**

As it can be seen, non-exhaustive search significantly decreases the number of comparisons, with a negligible drop in the performance. For example, for $W = 32$, almost exhaustive search performance is achieved for 25 times faster search. The overall search time is proportional to the number of comparisons, as long as the overhead of indexing is negligible compared to the overall search time. As the number of comparisons decreases, the search time also decreases. Hence the overhead is no longer negligible and the obtained speed-up is

less compared to the decrease in the number of comparisons.

We also compare our method with the state-of-the-art non-exhaustive search methods in the literature. This time, the performance of our method is tested on SIFT1B dataset [10], which consists of 1 Billion samples. We compare against the state-of-the-art indexing based non-exhaustive search method *Locally Optimized Product Quantization* (**LOPQ**) [14], *Inverted File System with Asymmetric Distance Calculation* (**IVFADC**) [10] and IVFADC adaptation of OPQ (**I-OPQ**) [12], [14]. The compared methods use 64-bits for the codes and 13-bits for the coarse quantizer ($K = 8192$), visiting the nearest 64 cells ($W = 64$). In order to use approximately the same amount of bits, we use 10 layers ($M = 10$) and the first two layers are used for indexing. The compared methods visit 64 cells among 8192, which is approximately the same as visiting 22 cells for two layers of 256, hence we select $W = 22$ ($64/8192 \cong 22^2/256^2$). The results are presented in **TABLE 7**. Here note that, the total length of the binary code is not different in non-exhaustive search tests, unlike the most common cases in the literature, where the coarse quantizer indexes are calculated as an extra cost, i.e., [10], [12], [14], [16], [26], [27].

TABLE 7
RECALL@R RESULTS FOR SIFT1B

	#bits	recall@1	recall@10	recall@100
IVFADC	77	0.088	0.372	0.733
I-OPQ	77	0.114	0.399	0.777
LOPQ	77	0.199	0.586	0.909
CompQ	80	**0.222**	**0.626**	**0.914**

As it can be seen, our algorithm outperforms the state-of-the-art methods for all three scores, using 3 more bits per sample. This corresponds to an increase in storage around 3.9% only.

4 Discussions

As explained in detail in **Section 2**, *CompQ* presents a quantization scheme based on the addition of several codevectors. A joint optimization scheme is proposed on top of a hierarchical structure. This structure is similar to the hierarchy of RVQ and its variants, but *CompQ* outperforms these methods thanks to the jointly computed codebooks. Other state-of-the-art methods such as OCKM and AQ also propose a joint optimization for codebook generation, but the lack of codebook hierarchy leads to more complex search spaces for encoding. OTQ and CQ instead, limit the search space with stronger constraints, decreasing the search complexity but resulting in inferior codebooks. *CompQ* is a viable compromise, where the search complexity is reduced by adopting a hierarchical structure while the codebook generation is improved with the proposed joint optimization scheme.

4.1 Joint Optimization

Investigating the behavior of the quantization errors in each layer is a good way to visualize the advantage of joint optimization against training all codebooks separately. As it can be seen in **Fig. 2**, the first layers reduce the quantization error the most, while the contributions of the last layers are more limited. It is worth mentioning that, obtaining a perfectly equal distribution of quantization error cannot be expected. Because at each layer, the residuals from the previous layers are quantized, hence the norms are decreased. However, a better distribution of the workload between the layers is shown to be possible with the proposed joint optimization scheme.

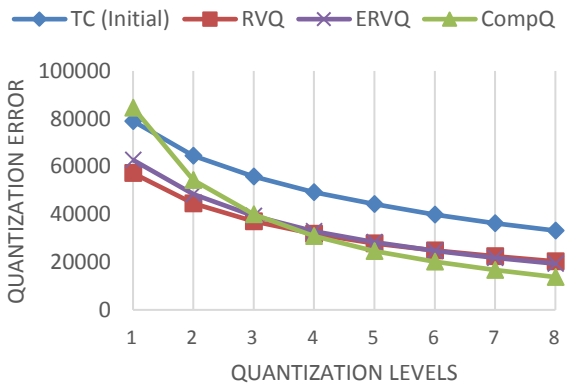

Fig. 2: Quantization error on SIFT1M as a function of the quantization levels for 64-bit encoding.

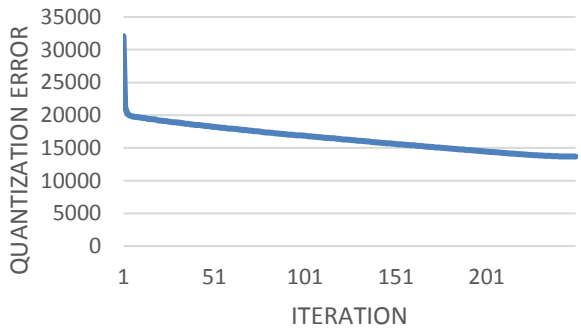

Fig. 3 Quantization error on SIFT1M as a function of training iterations for 64-bit encoding.

Fig. 3 shows the decrease in the quantization error for 250 iterations. As it can be seen, there is a steep drop right after the first iteration, then the error keeps decreasing slowly and converges around 13700. That also shows that the proposed initialization prevents overfitting and allows further iterations to improve the quantization performance.

4.2 Impact of the Proposed Novelties on the Performance

In **TABLE 8**, the impact of the proposed novelties on the performance are individually presented. The joint optimization scheme for codebook generation is compared to codebook generation scheme of RVQ. Also the winner codevector improvement is tested with RVQ and the performance is compared with *CompQ*. Both novelties are shown to provide significant improvement.

TABLE 8
IMPACT OF THE PROPOSED NOVELTIES ON THE PERFORMANCE SIFT1M, 64-BIT CODES

	MSE_Q	recall@1	recall@10	recall@100
RVQ	20302.1	0.257	0.659	0.952
CompQ H=1	17765.9	0.286	0.709	0.966
RVQ H=8	18735.3	0.298	0.726	0.973
CompQ H=8	14418.1	0.339	0.783	0.983
CompQ H=16	13964.2	0.347	0.786	0.986
CompQ H=32	**13671.2**	**0.352**	**0.795**	**0.988**

4.3 Parameter Selection

CompQ consists of several parameters, such as the number of quantization levels M, the number of codevectors per layer K, the learning rate γ, and the number of candidates for the winner codevector, H. For M and K we follow the same settings as in the literature, i.e., $M = 8$ for 64-bits code and $M = 4$ for 32-bits code. Similarly, we set $K = 256$.

Selecting a suitable number for the candidates of winner codevectors is a tradeoff between the encoding complexity and the quantization performance. The quantization performance increases with H and so as the complexity. Hence we select $H = 32$ since it gives the comparable encoding complexity with the competeing methods as shown in **TABLE 10**. We present the performance of our method for different values of H in **TABLE 8**. with codebooks trained with RVQ.

Note that when $H = 1$, the encoding scheme is the same as in RVQ. **TABLE 8** also shows the increase in the performance when our encoding scheme is used. For the learning rates, since the upper layers correspond to greater quantization errors, the weight of the corresponding update should be also greater. In order to do that, we select the corresponding learning rate of each layer using the equation in (26).

$$\gamma_m = \frac{1}{\lceil \log_2 m + 1 \rceil} \gamma_0 \tag{26}$$

Then we define a total value $\gamma_{Total} = \sum \gamma_m$ and normalize γ_m so that γ_{Total} is equal to the predefined value. The quantization performance for different values of γ_{Total} is presented in **Fig. 4**.

As it can be seen, there is no significant change in the quantization error for the tested values of γ_{Total} and hence we pick $\gamma_{Total} = 0.5$ which corresponds to a slightly improved quantization error according to our simulations.

Fig. 4: Quantization error on SIFT1M as a function of the learning rate parameter γ_{Total} for 64-bit encoding, H=1.

4.4 Initialization Methods

The importance of initialization for the proposed codebook training method is tested using different initialization schemes. Randomly initialized codebooks, RVQ initialized codebooks and the proposed codebook initialization, which is based on TC, are compared and the results are presented in **TABLE 9**.

TABLE 9
COMPARISON OF INITIALIZATION METHODS, SIFT1M, 64-BIT CODES, H=1

	MSE_Q	recall@1	recall@10	recall@100
CompQ Rand Init	21568	0.227	0.614	0.931
CompQ RVQ Init	17823	0.278	0.704	0.963
CompQ TC Init	**17765**	**0.286**	**0.709**	**0.966**

As it can be seen, random initialization provides inferior results as some codevectors are not updated during the training procedure. RVQ initialization is also outperformed by the proposed TC initialization, as the former overfits the training data at each layer while the latter provides a more balanced distribution of codevectors throughout the feature space.

4.5 Computational Complexity Analysis

The computational cost of encoding for *CompQ* can be calculated as in (22). The first term in (22) is the distance of the given query to the codevectors of the previous layer, so at this layer no extra calculations are required. The third and fourth terms are obtained using a look-up table. This requires mKH look-ups and additions for the m^{th} layer. The second term is the dot product of the given query with each codevector of the current layer, which costs $O(DK)$. Finally, among the distances calculated between all codevectors and the previous residual candidates, the best H are selected. This operation costs $O(KH \log H)$. These calculations are repeated for all M layers, and the code corresponding to the best quantization error is returned. The final cost can be expressed as,

$$O\left(MDK + \frac{(M-1)(M-2)}{2} KH + MKH \log H\right) \tag{27}$$

As it can be seen in **TABLE 10**, the computational cost of our method is comparable to the other methods, but significantly lower than AQ/APQ. A detailed analysis of the storage requirements is also presented in **TABLE 10**. *CompQ* requires to store K vectors of dimension D for each of the M layers, resulting in a storage cost of $O(MDK)$, which corresponds to $2MB$ for the SIFT dataset for 64-bits encoding. It can be observed from the table that the storage requirement of our method is also comparable with the other methods.

4.6 Relations with the Enhanced Residual Vector Quantization Method

ERVQ [17] is an extension over RVQ [16], which aims to improve the quantization quality by an iterative enhancement process over RVQ's training scheme. ERVQ starts with an RVQ initialization, and after that while keeping $M - 1$ codebooks fixed, it recalculates the codebook of the M^{th} layer. Several iterations are performed until convergence is reached.

TABLE 10
COMPARISON OF COMPUTATIONAL AND STORAGE COSTS

Method	Cost of Encoding	Cost of Encoding for Different Datasets and Code Lengths			
		SIFT1M-32	SIFT1M-64	GIST1M-32	GIST1M-64
RVQ/ERVQ	$O(MKD)$	131072	262144	983040	1966080
CKM/OPQ	$O(D^2 + KD)$	**49152**	**49152**	1167360	1167360
OCK	$O(TKD)$	327680	327680	2457600	2457600
AQ	$O(M^2K^2(M + \log(MK)) + KD)$	14712832	79724544	14925824	79937536
APQ	$O\left(D^2 + \frac{M}{4}(4^2K^2(4 + \log(4K)) + KD)\right)$	14729216	14761984	15847424	16093184
CQ	$O(3MKD)$	393216	786432	2949120	5898240
OTQ	$O(D^2 + KD + MK^2)$	311296	573440	1429504	1691648
KSSQ	$O(KD + 2\mathcal{K}DL + 8\mathcal{K}M)$	115200	197632	**338176**	**645632**
CompQ	$O\left(MDK + \frac{(M-1)(M-2)}{2}KH + MKH\log H\right)$	319488	761856	1171456	2465792

Method	Storage Cost	Storage Costs for Different Datasets and Code Lengths (MB)			
		SIFT1M-32	SIFT1M-64	GIST1M-32	GIST1M-64
RVQ/ERVQ	$O(MKD)$	1.00	2.00	7.5	15
CKM/OPQ	$O(D^2 + KD)$	**0.38**	**0.38**	8.91	**8.91**
OCK	$O(D^2 + 2KD)$	0.63	0.63	10.78	10.78
AQ	$O(MKD)$	1.00	2.00	7.50	15.00
APQ	$O(D^2 + MKD)$	1.13	2.13	14.53	22.03
CQ	$O(MKD)$	1.00	2.00	7.50	15.00
OTQ	$O(D^2 + MKD)$	1.13	2.13	14.53	22.03
KSSQ	$O(KDL)$	5.00	10.00	**4.69**	9.38
CompQ	$O(MKD)$	1.00	2.00	7.50	15.00

***K*: number of sub-codewords**	256	256	256	256
***D*: number of dimensions**	128	128	960	960
***M*: number of sub-codebooks**	4	8	4	8
***T*: search depth for OCK**	10	10	10	10
***H*: number of winner candidates**	32	32	32	32
$\mathcal{K}$: number of selected subspaces for KSSQ encoding	16	16	8	8
***L*: number of reduced dimensions (on average) for KSSQ**	20	40	20	40

Compared to ERVQ, the proposed method has three significant differences. The first one is the proposed initialization scheme, which is based on TC [11] instead of RVQ. **TABLE 9** shows the improvement in the performance provided by the TC based initialization method. The second difference is the training scheme. While ERVQ can only update one codebook per iteration, in the proposed method, all M codebooks are updated according to the quantization error, as explained in **Section 2.1**, using the formula given in (18). The contribution of the proposed training method on RVQ is shown in **TABLE 8**. As it can be seen, the proposed method already performs better than ERVQ with these two improvements. The third and the last difference between the proposed method and ERVQ is the encoding scheme. The contribution of the proposed encoding scheme is also shown in in **TABLE 8**. Combining these three new features, the proposed method outperforms ERVQ with an important margin (a relative improvement of 27.5% for SIFT1M and 34.8% for GIST1M), clearly demonstrating the significance of the proposed contributions.

5 CONCLUSIONS

In this paper, a novel vector quantization method based on the addition of codevectors is presented. A novel training algorithm, which minimizes the quantization error using a joint optimization among different codebook layers is proposed. The proposed method, *CompQ*, also improves the traditional encoding scheme of RVQ by redefining the winner codevector. By means of such novel improvements, *CompQ* achieves the state-of-the-art performance with comparable computational and storage costs. In the future, we aim to test the performance of *CompQ* for different distance metrics and for k-Nearest Neighbor classification problems.

REFERENCES

[1] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for Similarity Search: A Survey," in *arXiv preprint*, 2014, p. :1408.2927.

[2] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-Sensitive Hashing Scheme Based on P-

stable Distributions," in *SCG*, 2004, p. 253.
[3] X. He, D. Cai, S. Yan, and H. Zhang, "Neighborhood Preserving Embedding," in *ICCV*, 2005.
[4] Y. Weiss, A. Torralba, and R. Fergus, "Spectral Hashing," in *NIPS*, 2009, pp. 1753–1760.
[5] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognit. Lett.*, vol. 31, no. 11, pp. 1348–1358, Aug. 2010.
[6] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *CVPR*, 2011, pp. 817–824.
[7] D. Zhang, J. Wang, D. Cai, and J. Lu, "Self-Taught Hashing for Fast Similarity Search," in *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010, pp. 18–25.
[8] G. Lin, C. Shen, D. Suter, and A. van den Hengel, "A General Two-Step Approach to Learning-Based Hashing," in *ICCV*, 2013, pp. 2552–2559.
[9] A. Gordo, F. Perronnin, Y. Gong, and S. Lazebnik, "Asymmetric distances for binary embeddings.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 1, pp. 33–47, Jan. 2014.
[10] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–28, Jan. 2011.
[11] J. Brandt, "Transform coding for fast approximate nearest neighbor search in high dimensions," in *CVPR*, 2010, pp. 1815–1822.
[12] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized Product Quantization.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 4, pp. 744–755, Dec. 2014.
[13] M. Norouzi and D. J. Fleet, "Cartesian K-Means," in *CVPR*, 2013, pp. 3017–3024.
[14] Y. Kalantidis and Y. Avrithis, "Locally Optimized Product Quantization for Approximate Nearest Neighbor Search," in *CVPR*, 2014.
[15] E. C. Ozan, S. Kiranyaz, and M. Gabbouj, "K - Subspaces Quantization for Approximate Nearest Neighbor Search," *IEEE Trans. Knowl. Data Eng.*, 2016.
[16] Y. Chen, T. Guan, and C. Wang, "Approximate nearest neighbor search by residual vector quantization," *Sensors*, vol. 10, no. 12, pp. 11259–11273, 2010.
[17] L. Ai, J. Yu, Z. Wu, Y. He, and T. Guan, "Optimized residual vector quantization for efficient approximate nearest neighbor search," *Multimed. Syst.*, Jun. 2015.
[18] B. Wei, T. Guan, and J. Yu, "Projected Residual Vector Quantization for ANN Search," *IEEE Multimed.*, vol. 21, no. 3, pp. 41–51, Jul. 2014.
[19] A. Babenko and V. Lempitsky, "Additive Quantization for Extreme Vector Compression," in *CVPR*, 2014, pp. 931–938.
[20] J. Wang, J. Wang, J. Song, X.-S. Xu, H. T. Shen, and S. Li, "Optimized Cartesian K-Means," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 180–192, Jan. 2015.
[21] T. Zhang, D. Chao, and J. Wang, "Composite Quantization for Approximate Nearest Neighbor Search," in *ICML*, 2014.
[22] A. Babenko and V. Lempitsky, "Tree Quantization for Large-Scale Similarity Search and Classification," in *CVPR*, 2015.
[23] J. a Hertz, A. S. Krogh, R. G. Palmer, and A. S. Weigend, *Introduction to the Theory of Neural Computation*. 1993.
[24] W. Dong, M. Charikar, and K. Li, "Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces," in *SIGIR*, 2008, p. 123.
[25] J.-P. Heo, Z. Lin, and S.-E. Yoon, "Distance Encoded Product Quantization," in *CVPR*, 2014, pp. 2139–2146.
[26] A. Babenko and V. Lempitsky, "The inverted multi-index," in *CVPR*, 2012, vol. 14, no. 1–3, pp. 3069–3076.
[27] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: Re-rank with source coding," *ICASSP*, no. 3, pp. 861–864, 2011.
[28] J. Song, H. T. Shen, J. Wang, Z. Huang, N. Sebe, and J. Wang, "A Distance-Computation-Free Search Scheme for Binary Code Databases," *IEEE Trans. Multimed.*, vol. 18, no. 3, pp. 484–495, 2016.
[29] A. Punjani and D. J. Fleet, "Fast Search in Hamming Space with Multi-Index Hashing Mohammad Norouzi," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 6, pp. 1107–1119, 2014.

Ezgi Can Ozan received his BS degree in Electrical and Electronics Department at Middle East Technical University, Ankara, Turkey, in 2007 and MS degree in Signal Processing from the same University, in 2011. He is currently a Ph.D. candidate and working as a researcher at Tampere University of Technology, Tampere, Finland. His areas of interest include large-scale multimedia search, pattern recognition, machine learning, and computer vision.

Serkan Kiranyaz received his BS degree in Electrical and Electronics Department at Bilkent University, Ankara, Turkey, in 1994 and MS degree in Signal and Video Processing from the same University, in 1996. He worked as a Senior Researcher in Nokia Research Center and later in Nokia Mobile Phones, Tampere, Finland. He received his PhD degree from Tampere University of Technology; Institute of Signal Processing in 2005 and his Docency at 2007 respectively. He is currently working as a Professor in Electrical Engineering Department of Qatar University. Prof. Kiranyaz published 2 books, more than 35 journal papers on several IEEE Transactions and some other high impact journals and 80+ papers in international conferences. His recent publication, "Automatic Object Segmentation by Quantum Cuts" won the IBM Best Paper Award in ICPR'14. Prof. Kiranyaz is a senior of IEEE.

Moncef Gabbouj received his BS degree in electrical engineering in 1985 from Oklahoma State University, Stillwater, and his MS and PhD degrees in electrical engineering from Purdue University, West Lafayette, Indiana, in 1986 and 1989, respectively. Dr. Gabbouj is a Professor of Signal Processing at the Department of Signal Processing, Tampere University of Technology, Tampere, Finland. He was Academy of Finland Professor during 2011-2015. He held several visiting professorships at different universities. His research interests include multimedia content-based analysis, indexing and retrieval, machine learning, nonlinear signal and image processing and analysis, voice conversion, and video processing and coding. Dr. Gabbouj is a Fellow of the IEEE and member of the Academia Europa and the Finnish Academy of Science and Letters. He is the past Chairman of the IEEE CAS TC on DSP and committee member of the IEEE Fourier Award for Signal Processing. He served as Distinguished Lecturer for the IEEE CASS. He served as associate editor and guest editor of many IEEE, and international journals. Dr. Gabbouj was the recipient of the 2015 TUT Foundation Grand Award, the 2012 Nokia Foundation Visiting Professor Award, the 2005 Nokia Foundation Recognition Award, and several Best Paper Awards. He published over 650 publications and supervised 40 doctoral theses.

VI

A VECTOR QUANTIZATION BASED K-NN APPROACH FOR LARGE-SCALE IMAGE CLASSIFICATION

by

E.C.Ozan, E.Riabchenko, S. Kiranyaz and M. Gabbouj, 2016

Sixth International Conference on Image Processing Theory, Tools and Applications (IPTA), Oulu, 2016, pp. 1-6.

A Vector Quantization Based k-NN Approach for Large-Scale Image Classification

Ezgi Can Ozan[1], Ekaterina Riabchenko[1], Serkan Kiranyaz[2] and Moncef Gabbouj[1]
[1]Department of Signal Processing, Tampere University of Technology, Tampere, Finland
e-mail: {ezgi.ozan, ekaterina.riabchenko, moncef.gabbouj}@tut.fi
[2]Electrical Engineering Department, College of Engineering, Qatar University, Qatar
e-mail: mkiranyaz@qu.edu.qa

***Abstract*— The k-nearest-neighbour classifiers (k-NN) have been one of the simplest yet most effective approaches to instance based learning problem for image classification. However, with the growth of the size of image datasets and the number of dimensions of image descriptors, popularity of k-NNs has decreased due to their significant storage requirements and computational costs. In this paper we propose a vector quantization (VQ) based k-NN classifier, which has improved efficiency for both storage requirements and computational costs. We test the proposed method on publicly available large scale image datasets and show that the proposed method performs comparable to traditional k-NN with significantly better complexity and storage requirements.**

***Keywords*— k-NN Classifier; Vector Quantization; Large-Scale Image Classification.**

I. INTRODUCTION

The Nearest Neighbour classification is one of the most popular methods in data mining [1]. This idea is very simple and easy to envision. Many statistical classification methods aim to estimate the underlying model which generated the corresponding sample set. In most cases, the only information about this model is inferred from the set of samples at hand. Hence, this may not be a simple task, as the obtained model strictly depends on the previously made assumptions. However, it is always reasonable to assume that, samples which are close to each other by an appropriate distance metric, also have the same class labels. In other words, if there are enough reference samples at hand and the reference set is a good representation of the test set, then the test samples can be classified according to the nearest reference samples. This intuitive method is called the Nearest Neighbour (*1-NN*) classification [2]. It classifies a given input according to the class of its nearest neighbour, among a stored set of reference samples.

Obviously, 1-NN algorithm is highly susceptible to noise, as the classification decision is based on only one reference sample. In order to improve the robustness against the noise, the main idea behind the *1-NN* can be naturally extended to *k-NN*, where *k* nearest neighbours are found and the classification decision is given after a majority voting among the obtained nearest reference samples [3]. A comparison of 1-NN and k-NN on a toy dataset is shown in **Fig. 1**.

One can also suggest that, the reference samples which are closer to the test sample should have a higher impact on the class decision. This idea leads to the *weighted k-NN*, where the votes of the reference samples are weighted according to their distance to the test sample. Dudani proposes several weighting schemes in [4] and shows that the weighted k-NN outperforms the majority voting based k-NN. After Dudani, Pao *et al.* uses Fibonacci series as the weighting function in [5]. They show that Fibonacci weighted voting for k-NN outperforms the linear mapping proposed by Dudani. Gou *et al.* extends the linear mapping of Dudani with the reciprocal of the ranking and proposes a new weighting scheme in [6]. Gou *et al.* also proposes another method in [7], which adds nonlinearity to Dudani's method and shows that his method outperforms 1-NN, majority voting and Dudani's linear weighting methods.

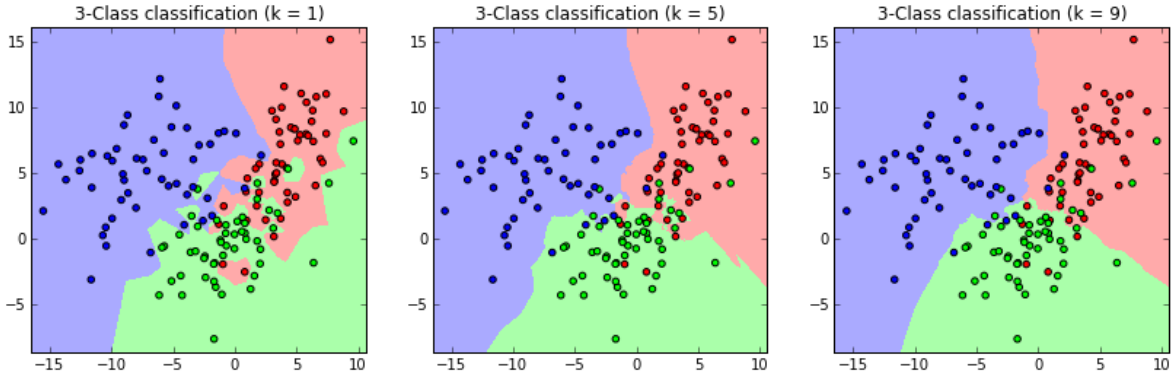

Fig. 1. 1-NN vs k-NN on a toy dataset.

One major drawback of NN based algorithms is that, they require to store the whole reference set. Learning based classification algorithms aim to estimate the parameters using the given training set in order to obtain a statistical model, while as an instance based classification algorithm, NN stores the whole training set. Besides the storage problem, NN based methods are computationally very complex, as retrieval of the nearest neighbours for a given sample is linearly proportional to both the size and the cardinality of the training set [1]. As the number of samples and their dimensions increase, the computational and storage requirements also increase, hence the popularity of NN based methods decrease due to this practical challenges.

In this paper, we propose a *Vector Quantization* (*VQ*) based k-NN solution to the image classification problem. *Vector Quantization for Approximate Nearest Neighbour Search* (VQ-ANN) is a well-studied field, which aims to compress vectors with high number of dimensions into binary strings. With this compression, both the storage space requirements and computational costs are significantly decreased [8]. In the proposed method, we show that, a more efficient k-NN classifier can be obtained by the help of the aforementioned VQ-ANN techniques. The proposed classifier uses approximated distances between the test sample and the reference set, resulting in significantly better computational costs and storage requirements, while having a comparable classification performance.

The rest of the paper is organized as follows. In **Section II**, problem formulation is presented and background information for the terms used in the proposed method are presented. In **Section III**, the proposed method is explained more in detail. **Section IV** presents the experiments and finally in **Section V** the paper is concluded.

II. PROBLEM FORMULATION

In this section, we present the formulation for the k-NN problem and provide some background information about the terms used in the proposed method.

A. The Approximate Nearest Neighbour Search

Given a sample $\boldsymbol{y} \in \mathbb{R}^D$ and a set of N samples $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\} \in \mathbb{R}^{D\times N}$, the nearest neighbor search aims to find the nearest sample for $\boldsymbol{y}$ among $\boldsymbol{X}$ as

$$\overline{\boldsymbol{x}} = \underset{\boldsymbol{x}_i \in \boldsymbol{X}}{\operatorname{argmin}}\, d(\boldsymbol{y}, \boldsymbol{x}_i) \tag{1}$$

where $d(\boldsymbol{y}, \boldsymbol{x}_i)$ is the distance between $\boldsymbol{y}$ and $\boldsymbol{x}_i$.The nearest neighbour search can be extended to k-Nearest Neighbour search by simply retrieving the first k samples as

$$\overline{\boldsymbol{x}}_k = \underset{\boldsymbol{x}_i \in (\boldsymbol{X} / \cup_{m=1}^{k-1} \overline{\boldsymbol{x}}_m)}{\operatorname{argmin}} d(\boldsymbol{y}, \boldsymbol{x}_i). \tag{2}$$

The nearest neighbour search can be accelerated using an approximate method to calculate the distance $d(\boldsymbol{y}, \boldsymbol{x}_i)$. This is called the Approximate Nearest Neighbor (ANN) search. Normally, the distance $d(\boldsymbol{y}, \boldsymbol{x}_i)$ between the samples y and x_i is calculated for every $\boldsymbol{x}_i \in \boldsymbol{X}$. The complexity of this operation is linearly dependent on the number of dimensions D and the number of samples N. In order to approximate the distances, ANNs use the quantized versions of reference samples. For a given sample $\boldsymbol{x}_i$, let the quantizer $\mathbb{Q}$ quantize $\boldsymbol{x}_i$ to be a set of codevectors $\boldsymbol{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2 \dots \boldsymbol{v}_H\}$ as

$$\widehat{\boldsymbol{x}}_i = \mathbb{Q}(\boldsymbol{x}_i) \quad \widehat{\boldsymbol{x}}_i \in V, \tag{3}$$

meaning that there are H different codevectors, so once the distance between the test sample y and these codevectors are calculated, then the approximated distance $\hat{d}(\boldsymbol{y}, \boldsymbol{x}_i) = d(\boldsymbol{y}, \widehat{\boldsymbol{x}}_i)$ can be calculated using a look-up table.

B. The Nearest Neighbour Classification

Given a sample $\boldsymbol{y} \in \mathbb{R}^D$, the set of N reference samples $\overline{\boldsymbol{X}} = \{\overline{\boldsymbol{x}}_1, \overline{\boldsymbol{x}}_2, \dots, \overline{\boldsymbol{x}}_N\} \in \mathbb{R}^{D\times N}$ is the sorted set of reference samples according to the distance, i.e., $d(\boldsymbol{y}, \overline{\boldsymbol{x}}_i) \leq d(\boldsymbol{y}, \overline{\boldsymbol{x}}_j) \Leftrightarrow i < j$. Also given a corresponding label set $\boldsymbol{L} = \{l_1, l_2, \dots, l_N\} \in \mathbb{R}^{1\times N}$, where $l_i \in \{1, 2, \dots C\}$, l_i shows the label of sample vector $\overline{\boldsymbol{x}}_i$. 1-NN classifier $\mathbb{C}_{1NN}$ classifies the given sample $\boldsymbol{y}$ as given below:

$$\mathbb{C}_{1NN}(\boldsymbol{y}) = l_1. \tag{4}$$

Similarly a majority voting k-NN classifier $\mathbb{C}_{kNN}$ classifies the given sample y to its corresponding class as follows:

$$\mathbb{C}_{kNN}(\boldsymbol{x}) = \underset{c}{\operatorname{argmax}} \sum_{k=1}^{K} \Theta(c = l_k), \tag{5}$$

where Θ is the indicator function defined as follows:

$$\Theta(a) = \begin{cases} 1 & a = true \\ 0 & a = false \end{cases}. \tag{6}$$

Furthermore, a weighted k-NN classifier $\mathbb{C}_{wkNN}$ classifies the given sample $\boldsymbol{y}$ to its corresponding class as given below:

$$\mathbb{C}_{kNN}(\boldsymbol{x}) = \underset{c}{\operatorname{argmax}} \sum_{k=1}^{K} w_k \Theta(c = l_k), \tag{7}$$

where w_k is the corresponding weight for the k^{th} neighbor.

As explained above, for k-NN classification, a nearest neighbour search is required for every given test sample, meaning that the classification of a given sample requires N distance calculations, each of them is linearly dependent on the number of dimensions D. This procedure is computationally very expensive, especially for large values

of N and D. Besides, k-NN requires a reference sample set to be stored. The size of this set is also proportional to N and D. As the number of samples and the dimensions increase, k-NN becomes infeasible. In this paper we propose to introduce vector quantization methods to k-NN classifiers in order to approximate the distance calculation, and store the reference samples as a set of compressed binary strings. In the next section, we explain the proposed method more in detail.

III. PROPOSED METHOD

In this section, we explain the details of the proposed method. In our method, we aim to overcome the main disadvantages of k-NN classifiers, by introducing VQ based ANNs. As mentioned above, for a given test sample, a k-NN classifier first finds the nearest k reference samples from a stored set. Then using a voting approach, the decision is made for the classification of the given sample. In our method, we propose approximate nearest neighbour search to find the nearest k samples.

A. Residual Vector Quantization for ANN

There are many different VQ methods in the literature [8], and in our method we propose to use *Residual Vector Quantization for Approximate Nearest Neighbour Search* (RVQ) [9]. Residual vector quantization is a well-studied quantization scheme [10] and it has been adapted to ANN by Chen *et al.* [9].

RVQ performs quantization using a layer based hierarchical structure. As the name implies, the quantization is performed on the residuals of the previous layer. The final quantized vector is obtained by summation of the codevectors of each layer. For an M layer RVQ, the quantization process can be formulized as:

$$\begin{aligned} MSE_Q^{(RVQ_1)} &= \min_{\{\boldsymbol{V}_1\},\{\boldsymbol{B}_1\}} \frac{1}{N}\sum_{i=1}^{N}\left\|\left(\boldsymbol{x}_i - \sum_{m=1}^{1}\boldsymbol{V}_m\boldsymbol{b}_{m_i}\right)\right\|^2 \\ &\vdots \\ MSE_Q^{(RVQ_M)} &= \min_{\{\boldsymbol{V}_M\},\{\boldsymbol{B}_M\}} \frac{1}{N}\sum_{i=1}^{N}\left\|\left(\boldsymbol{x}_i - \sum_{m=1}^{M}\boldsymbol{V}_m\boldsymbol{b}_{m_i}\right)\right\|^2 \end{aligned} \tag{8}$$

Here $\boldsymbol{V}_m$ represents the codebook matrix for the layer m, and $\boldsymbol{b}_{m_i}$ is a binary selection vector, which selects only one codevector from the given codebook.

As it can be seen in above, RVQ separates the VQ problem into M sub-problems and solves it in a hierarchical way. RVQ requires a training stage in order to learn the codebooks. For each layer, one codebook is obtained using K-Means on the given set of training vectors. Then each sample vector is appointed to the nearest codevector, and the residuals are calculated. The residuals are then passed to the next layer and this process is repeated for all the layers. The encoding scheme proposed in RVQ [9] stems from the same idea used in the training stage. For a given novel sample, the nearest codevector of the first codebook is found, then the residual is calculated and passed to the next layer. Finally M codevectors, one from each layer, are obtained. Their indices for the corresponding layers are stored as binary strings. The length of this binary string is obtained as given as:

$$bitlength = M \log_2 \mathcal{K}, \tag{9}$$

where $\mathcal{K}$ represents the number of codevectors in a codebook.

The Euclidean distance between a given test sample and a compressed reference vector $d(\boldsymbol{y}, \hat{\boldsymbol{x}})$ is calculated as presented as:

$$\begin{aligned} \hat{\boldsymbol{x}} &= \sum_{m=1}^{M} \dot{\boldsymbol{v}}_m \\ d(\boldsymbol{y}, \hat{\boldsymbol{x}})^2 &= \|\boldsymbol{y}\|^2 - 2\sum_{m=1}^{M}\langle \boldsymbol{y}, \dot{\boldsymbol{v}}_m\rangle + \sum_{m=1}^{M}\sum_{l=1}^{M}\langle \dot{\boldsymbol{v}}_m, \dot{\boldsymbol{v}}_l\rangle \end{aligned} \tag{10}$$

Here $\dot{\boldsymbol{v}}_m$ is the codevector corresponding to the code at the M^{th} layer.

As it can be seen, if a look-up table for the dot-products of each codevector and the given test sample $\langle \boldsymbol{y}, \dot{\boldsymbol{v}}_m\rangle$ is prepared, the approximated distance can be calculated using $M^2/2 + M$ look-ups ($M^2 \ll N$). Note that, the dot-products of codevectors $\langle \dot{\boldsymbol{v}}_m, \dot{\boldsymbol{v}}_l\rangle$ are already calculated and stored in a look-up table in the training stage.

Another property of the layered structure of RVQ is that, it can inherently perform indexing, which is required for non-exhaustive search [9]. The first layers of RVQ can be used for indexing of the database elements so that nearest neighbour search is performed only on a small subset. This accelerates the search time significantly.

B. Voting Based Classification

After finding the (approximate) k-nearest neighbours, the next step is to define a voting scheme and classify the given sample. Apart from the basic majority voting, several different methods have been proposed for the weighting scheme [3]–[7]. Here we explain *Majority Voting* [2], *Dudani* [4] and *Dual Distance* [7] methods in detail.

1) Majority Voting: In this method [2], the k-nearest neighbours contribute to voting with equal votes. Their distance to the given test sample or their rank in the ordering does not affect the final result. The class which has the most votes is selected.

2) Dudani Weights: In [4], Dudani proposes to impose a weighting scheme for the votes of each neighbour, according to their distance to the given test sample. This weighting is linear, and the weight of each neighbour is linearly mapped between [0, 1]. This weighting function can be formulated as given below:

$$w_k = \begin{Bmatrix} \left(\dfrac{d_K - d_k}{d_K - d_1}\right) & d_K \neq d_1 \\ 1 & d_K = d_1 \end{Bmatrix}, \tag{11}$$

where d_k is the distance between the given sample $\boldsymbol{y}$ and the k^{th} nearest neighbor $\bar{\boldsymbol{x}}_k$, i.e., $d_k = d(\boldsymbol{y}, \bar{\boldsymbol{x}}_k)$.

3) Dual Distance Weights: The *Dual Distance* method is proposed by Gou in [7]. In this weighting scheme, the contributions of the neighbours to the class decision is weighted as follows:

$$w_k = \begin{cases} \left(\frac{d_K - d_k}{d_K - d_1}\right)\left(\frac{d_K + d_1}{d_K + d_k}\right) & d_K \neq d_1 \\ 1 & d_K = d_1 \end{cases}. \qquad (12)$$

As it can be seen, Dual Distance is a modified version of Dudani's weighting scheme, where the mapping of distances to the interval [0, 1] is no longer linear.

IV. EXPERIMENTS

We evaluate the performance of the proposed method on three publicly available image datasets. The datasets are obtained from [11], [12]. We use PascalVOC07 and ImageNet datasets with "*decaf7*" features, and Bing dataset with "*decaf6*" features. We use the 25% of each dataset as the holdout test set. The "*decaf*" features are extracted from layers of a convolutional neural network [13]. "*Decaf6*" is extracted from the 6th layer and "*decaf7*" is extracted from the 7th layer. The details of the datasets used in the experiments is presented in in **Table 1**.

Table 1. Properties of the Selected Datasets

Property	Pascal	Bing	ImageNet
#Classes	20	257	118
#Training Samples	10740	90692	124200
#Test Samples	3580	30230	41400
#Dimension	4096	4096	4096
Feature Type	*decaf7*	*decaf6*	*decaf7*

Here we present the VQ based k-NN classifier results using three different voting schemes mentioned above. We compare these results to the traditional k-NNs using the corresponding voting scheme. In these experiments, SVMs serve as a baseline, thus we use the Python scikit-learn [14] wrapper for libSVM [15] with default parameters. In order to obtain the best performance with SVMs, parameters should be optimized for each dataset separately. The performance metric used for evaluating the classification accuracy is defined as given below:

$$accuracy = \frac{\#CorrectlyClassifiedSamples}{\#Samples} \qquad (13)$$

The results for the datasets Pascal, Bing and ImageNet are presented in **Table 2**. The majority voting method is denoted as *(MV)*, Dudani weighting scheme as *(D)* and Dual Distance weighting scheme as *(DD)*. As it can be seen, the VQ based k-NN performs comparable to the traditional k-NN. For Bing and ImageNet, weighted distance based k-NN methods are shown to perform better than majority voting, however for a smaller and noisier dataset Pascal, majority voting k-NN method is significantly better than weighted distance based methods.

k-NN methods are shown to outperform SVM on the Pascal, while SVM performs better on the ImageNet. For the Bing dataset, the obtained performances are comparable. Note that, SVMs or any learning based classifier can exploit the correlations among the dimensions of the feature vector, while k-NN classifiers require a proper distance metric to be defined beforehand. In our experiments, we use the Euclidean distance, yet any other distance metric can also be used. Furthermore, this distance metric can also be learned for a given dataset [16], but selection of the proper distance metric is beyond the scope of our paper.

Table 2. Classification accuracies obtained on different datasets, 128-bit encoding, K=64

Method	Pascal	Bing	ImageNet
k-NN (MV)	0.520	0.255	0.645
k-NN (D)	0.406	0.293	0.674
k-NN (DD)	0.381	0.294	0.675
VQ k-NN (MV)	0.532	0.241	0.647
VQ k-NN (D)	0.471	0.264	0.667
VQ k-NN (DD)	0.451	0.265	0.668
SVM	0.363	0.260	0.701

The effect of K on the performance of a VQ based k-NN classifier is represented in **Fig. 2**. As the value of K increases, the accuracy also increases up to a point. After that the accuracy starts to decline.

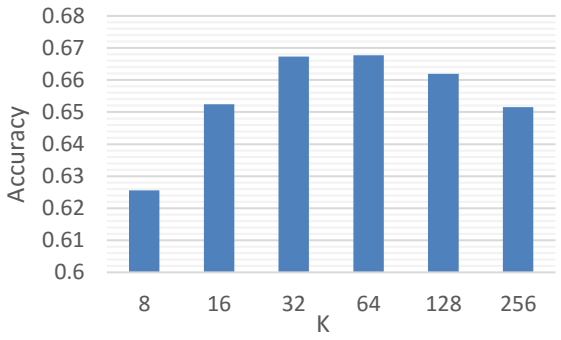

Fig. 2. Performance of VQ based k-NN using Dual Distance for different values of K on ImageNet Dataset (bit length = 128)

The effect of quantization bit length on the performance of the VQ based k-NN classifier is shown in **Fig. 3**. As the bit length increases, the accuracy also increases but together with the computational costs and storage requirements.

Table 3. Computational Costs and Storage Requirements

Complexity	Formulation	Pascal	Bing	ImageNet
k-NN	$ND + N\log_2 K$	44055480	372018584	509468400
VQ k-NN	$\left(\frac{M^2}{2} + M\right)\frac{N}{E} + \frac{N}{E}\log_2 K + 256MD$	16877904	17627454	17941591
SVM	$\left(\frac{C^2 - C}{2}\right)D$	778240	134742016	28274688
Storage	**Formulation**	**Pascal**	**Bing**	**ImageNet**
k-NN	ND	335 MB	2834 MB	3881 MB
VQ k-NN	$\frac{NM}{8} + 256MD$	2.3 MB	4.7 MB	5.8 MB
SVM	$\left(\frac{C^2 - C}{2}\right)D$	5.9MB	1028 MB	216 MB
Parameters		**Pascal**	**Bing**	**ImageNet**
C: #Classes		20	257	118
N: #Samples		10740	90692	124200
D: #Dimensions		4096	4096	4096
M: #RVQ Layers		16	16	16
K: #Nearest Neighbours		64	64	64
E: #Non-Exhaustive Search Constant		16	16	16

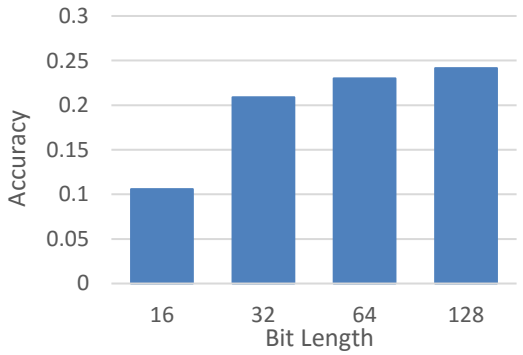

Fig. 3**.** Performance of VQ based k-NN using Majority Voting for different bit lengths on Bing Dataset (k=64)

The formulizations of computational cost and storage requirements are presented in **Table 3**. The traditional k-NN first calculates the Euclidean distances for the given N reference samples, then sorts the best K of them. VQ based k-NN performs the same operation but much faster. RVQ method requires $M^2/2 + M$ look-ups for distance calculation and thanks to the non-exhaustive search, it calculates only N/E distances. $256MD$ is the initialization cost of RVQ for a given sample [9]. SVM requires $(C^2 - C)/2$ dot-products to predict the class of a given sample [15]. As it can be seen, VQ approach significantly decreases the computational complexity of k-NN. Traditional k-NN requires the whole reference set to be stored, while the VQ based k-NN stores a compressed version of it. $256MD$ is the space required for the RVQ itself [9]. SVM requires to store $(C^2 - C)D/2$ dimensional hyperplanes [15]. The formulations use 8 bytes (double precision) as the unit memory.

V. CONCLUSION

In this paper, we introduce VQ-ANN search to k-NN classifiers for image classification. We show that, the two most significant drawbacks of k-NN classifiers, the storage requirement and computational cost, have been overcome. The experiments conducted on publicly available image datasets of different scales show that the proposed method provides comparable classification performance to traditional k-NN methods, with significantly lower computational costs and storage requirements, especially for large-scale datasets. We also compare our method with SVMs and show that the proposed method produces comparable results, with lower complexity and storage requirements on datasets with high number of classes. The gain in complexity and storage with respect to SVMs increases as the number of classes increase, keeping the number of samples in the reference set the same. However, the gain with respect to the traditional k-NNs increase as the number of reference samples increase, independent from the number of classes, until a convergence point is

reached. As a future work, we aim to introduce the class labels into the quantization in order to obtain a supervised quantization scheme, which is expected to improve the performance of the classifier.

REFERENCES

[1] X. Wu, V. Kumar, Q. J. Ross, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z. H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, *Top 10 algorithms in data mining*, vol. 14, no. 1. 2008.

[2] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[3] E. Fix and J. L. Hodges, "Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties," *Int. Stat. Rev. / Rev. Int. Stat.*, vol. 57, no. 3, p. 238, Dec. 1989.

[4] S. A. Dudani, "The Distance-Weighted k-Nearest-Neighbor Rule," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-6, no. 4, pp. 325–327, 1976.

[5] T.-L. Pao, Y.-T. Chen, J.-H. Yeh, Y.-M. Cheng, and Y.-Y. Lin, "A comparative study of different weighting schemes on KNN-based emotion recognition in Mandarin speech," *Adv. Intell. Comput. Theor. Appl. With Asp. Theor. Methodol. Issues*, pp. 997–1005, 2007.

[6] J. Gou, T. Xiong, and Y. Kuang, "A novel weighted voting for K-nearest neighbor rule," *J. Comput.*, vol. 6, no. 5, pp. 833–840, 2011.

[7] J. Gou, "A New Distance-weighted k -nearest Neighbor Classifier," *J. Inf. Comput. Sci.*, vol. 6, no. June, pp. 1429–1436, 2012.

[8] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for Similarity Search: A Survey," in *arXiv preprint*, 2014, p. :1408.2927.

[9] Y. Chen, T. Guan, and C. Wang, "Approximate nearest neighbor search by residual vector quantization," *Sensors*, vol. 10, no. 12, pp. 11259–11273, 2010.

[10] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Trans. Inf. Theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[11] T. Tommasi and T. Tuytelaars, "A Testbed for Cross-Dataset Analysis," in *Computer Vision - Eccv 2014 Workshops*, vol. 8927, 2015, pp. 18–31.

[12] T. Tommasi and T. Tuytelaars, "A Testbed for Cross-Dataset Analysis," *https://sites.google.com/site/crossdataset/home/files*, 2016. .

[13] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *ICML*, 2014, pp. 647–655.

[14] F. Pedregosa, O. Grisel, R. Weiss, A. Passos, and M. Brucher, "Scikit-learn : Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, no. 1, pp. 2825–2830, 2011.

[15] C.-C. Chang and C.-J. Lin, "Libsvm," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 1–27, 2011.

[16] K. Q. Weinberger and K. S. Lawrence, "Distance metric learning for large margin nearest neighbor classification," *J. Mach. Learn. Res.*, vol. 10, no. May, pp. 207–244, 2009.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-4012-7
ISSN 1459-2045