Learning compact representations for large scale image search

Himalaya Jain

ANNÉE 2018

UNIVERSITÉ DE RENNES 1

UNIVERSITE BRETAGNE LOIRE

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

Ecole Doctorale MathSTIC

présentée par

Himalaya Jain

préparée à l'unité de recherche Inria Rennes - Bretagne Atlantique
Institut National de Recherche en Informatique et Automatique
(Université de Rennes 1)

Learning compact representations for large scale image search

Thèse soutenue à Rennes
le date 4 Juin 2018

devant le jury composé de :

Matthieu Cord
Professeur, LIP6, Sorbonne Université, Paris. Rapporteur.

Florent Perronnin
Directeur Scientifique, NAVER LABS Europe, Grenoble. Rapporteur.

Christine Guillemot
Directrice de Recherche, Inria Rennes, Rennes. Examinatrice.

Frédéric Jurie
Professeur, Université Caen Normandie, Caen. Examinateur.

Hervé Jégou
Research Scientist, Facebook AI Research, Paris. Examinateur.

Ewa Kijak
Assoc. Professeur, Inria Rennes, Rennes. Examinatrice.

Rémi Gribonval
Directeur de Recherche, Inria Rennes, Rennes. Directeur de thèse.

Patrick Pérez
Directeur Scientifique, Valeo.ai, Paris. Co-directeur de thèse.

CONTENTS

ACKNOWLEDGEMENT

The time spent during my Ph.D. has been a great learning and fulfilling experience. Many have contributed directly or indirectly towards it.

Foremost, I would like to express my gratitude towards my supervisors Joaquin Zepeda, Patrick Pérez and Rémi Gribonval who have consistently given me support, encouragement and guidance for my research work. I thank Joaquin for many brainstorming sessions and feedback towards solving research problems, his support has been invaluable. I thank Patrick for his insightful feedback, guidance and, the support he gave during this time. Every discussion with him left me with a positive mindset and enthusiasm. I thank Rémi for helping me to be aware of the big picture and his thought-provoking advice and view of the problems we worked on.

I also want to thank Hervé Jégou for his contributions during our work on ECCV. Many thanks to my friends and labmates from Technicolor and Inria for their support and making my time at work more pleasant. I would also like to extend my thanks to all the reviewers and examiners for reading my manuscript and providing their feedback.

Most importantly, I want to thank my family for their unconditional love, support, and care. Special thanks to my brother, Mihir Jain, for his tremendous encouragement and guidance.

RÉSUMÉ

Le multiplication des appareils connectés a conduit à la production et au partage via Internet d'une quantité considérable de données. Il y a plus de 1,8 milliard de sites Web[1] accessibles par Internet, contenant des données sous forme de texte, d'images, de fichiers audio ou encore de vidéos. Cette grande quantité de données crée d'immenses opportunités, mais également d'important défis, en particulier relativement à leur compréhension, à leur analyse et à leur bonne organisation. Trouver des documents ou des informations intéressants dans cette quantité considérable de données, nécessite un système de recherche puissant et efficace. Pour concevoir un tel système, il est essentiel d'avoir un certain degré de compréhension du contenu et un moyen efficace de comparaison basé sur cette compréhension. Ces problèmes deviennent encore plus difficiles lorsque les données sont visuelles, parce que définir le sens ou le contenu de ce type de données ne se fait pas sans une certaine dose de subjectivité. Et donc, trouver une représentation capable de gérer les différentes significations sémantiques possibles des données est difficile. Dans notre travail, nous nous attachons à résoudre ce problème difficile à travers différentes contributions qui s'inscrivent dans la volonté de créer un *système de recherche d'images* efficace.

Un système de recherche d'images est censé indexer et encoder une large base de données de sorte que, lors d'une requête utilisateur, le système soit capable de récupérer les images pertinentes en temps réel. Comme nous le verrons dans les sections suivantes, un tel système repose sur une représentation précise et compacte des données.

Ce travail de thèse se concentre sur l'apprentissage de représentations d'images compactes et leur utilisation dans le cadre d'une recherche d'images efficace et précise. En particulier, il aborde la question de la recherche d'images à grande échelle.

Recherche d'images à grande échelle Les images sont présentes partout sur Internet et se multiplient à un rythme effréné. Par exemple, 147 000 images sont téléchargées chaque minute sur Facebook[2]. Cet accroissement exponentiel de la quan-

[1] http://www.internetlivestats.com/total-number-of-websites/
[2] https://sproutsocial.com/insights/facebook-stats-for-marketers/

tité de données visuelles, largement entraîné par la mise à disposition du grand public de caméras à bas prix, nécessite que nous développions des systèmes de recherche et de classification des images, afin d'augmenter la pertinence des contenus échangés. En outre, la recherche devrait être basée sur le contenu visuel des images, qui peuvent ou non bénéficier de métadonnées. Pour une comparaison précise des images, nous avons besoin d'une représentation d'image qui soit adaptée au type de comparaison désirée. D'importants efforts de recherche existent sur l'apprentissage de telles représentations, qu'elles reposent sur des descripteurs empiriques ou sur des réseaux de neurones convolutifs (convolution neural network, CNN). Les représentations d'images actuellement à la pointe de la recherche sont basées sur des CNN et offrent une excellente précision pour identifier le contenu visuel d'image (classification) ou pour comparer des images (recherche et fouilles basées sur la similarité). Ces représentations sont de grande dimension, typiquement 128 à 4096.

Lorsque la recherche se fait à grande échelle, il est nécessaire qu'elle soit efficace. L'efficacité repose sur la rapidité de comparaison des images, en limitant les accès disque qui sont coûteux, et en stockant les représentations vectorielles des images sur des mémoires rapides de type RAM.

Ainsi, pour une recherche à grande échelle, il est important d'avoir une représentation compressée des images, car cela permet au système d'avoir l'ensemble de la base de données dans la mémoire principale, évitant ainsi les va-et-vient suboptimaux entre le système et le disque dur. Pour nous rendre compte des avantages de la compression, considérons l'exemple suivant, qui porte sur la recherche d'images via un ordinateur portable. Supposons que nous ayons 8 Go de RAM disponible, notre mémoire ne peut contenir qu'un demi-million de vecteurs de dimension 4096, qui ont chacun besoin de 16Ko (en supposant des représentations en virgule flottante de 4 octets). Une représentation comprimée de 8 octets nous permet au contraire d'en stocker un milliard, soit 2000 fois plus. A noter qu'il est possible de stocker sur un serveur dans le cloud les images : tout ce dont nous avons besoin pour la recherche est la représentation compressée de ces images sur notre ordinateur portable. Une fois les images intéressantes trouvées, il ne nous restera plus qu'à récupérer ces images dans le serveur en linge.

Une autre exigence, essentielle, est que le calcul de la similarité/dissimilarité entre la requête et les points de la base de données soit efficace. De gros efforts de recherche ont été réalisés dans l'apprentissage des représentations d'images qui nous permettent de comparer deux images. Certaines de ces approches nécessitent également que ces représentations soient compactes; d'autres méthodes, au contraire – en particulier destinées à la recherche à grande échelle – comprennent une phase de réduction/compression dimensionnelle.

Dans la suite de ce résumé, nous discuterons brièvement de solutions qui pourraient être appliquées à la recherche à grande échelle, et présenterons nos contributions sur ce sujet.

Recherche des plus proches voisins

Supposons que nous ayons un excellent extracteur de caractéristiques pour les images, disons $f(.)$, et que, pour une image I_i, il donne un vecteur de dimension D, $x_i = f(I_i) \in R^D$. Supposons également que la similitude de contenu entre deux images est captée sous la forme d'une distance entre leurs caractéristiques respectives. Formellement, supposons que $d(x_i, x_j) < d(x_i, x_k)$ implique que I_i et I_j sont plus similaires que I_i et I_k, où $d(.,.)$ correspond à une mesure de distance, par exemple la distance euclidienne. Alors notre problème qui consiste à trouver des images similaires devient un problème de recherche des plus proches voisins dans l'espace des caractéristiques. Conséquence de la *malédiction de la dimension*, toute recherche du plus proche voisin ne peut être en pratique d'une complexité inférieure à l'ordre de grandeur de la base de données multiplié par D. Par conséquent, à grande échelle, la recherche exacte du voisin le plus proche prend énormément de temps et nécessite un important espace de stockage. Comme cela a été mentionné précédemment, il est important d'avoir une représentation compressée avec un moyen efficace de calculer la distance ou la similarité entre les caractéristiques. Donc, étant données les contraintes pratiques, faire un calcul de distance exact n'est pas réalisable ; il existe cependant des approches qui satisfont ces contraintes pratiques avec un calcul de distance approximatif. On parle alors de recherche approchée plus proches voisins (*approximate nearest neighbor search*, ANNs), i.e., la recherche d'éléments de la base qui ont une probabilité élevée d'être les plus proches voisins.

Les deux approches les plus populaires pour effectuer une recherche des ANNs sont la quantification vectorielle (*vector quantization*, VQ) et le hachage.

Dans les méthodes de hachage pour la recherche des ANNs, les éléments de la base sont d'abord associés à un code binaire en utilisant plusieurs fonctions de hachage. Ces fonctions sont censées préserver approximativement la relation de distance existant entre les éléments. La comparaison est réalisée à l'aide de la distance de Hamming entre les codes binaires, et est donc rapide à opérer.

L'autre approche pour la recherche des ANNs est la quantification vectorielle (VQ). C'est un moyen populaire de compresser les vecteurs. Dans le cadre de cette approche, les vecteurs sont approximativement représentés par un ou plusieurs vecteurs prédéfinis (*codeword*). Cette représentation approximative conduit à une forme compressée, étant donnée que chaque élément de la base est représenté par un ou quelques identifiants qui renvoient aux codeword(s) de code. Pour trouver les plus proches voisins d'une cible objet d'une requête, celle-ci est simplement comparée aux représentations approximatives des images de la base, ce qui fait que cette méthode est très efficace. Cette comparaison peut même être réalisée sans quantification de la requête; on appelle alors cette comparaison *calcul de distance asymétrique* (ADC). Comparée à une recherche avec requête quantifiée, ADC conduit à une meilleure approximation de la distance avec la même complexité, grâce à l'utilisation d'une *table de correspondance* (*l*ookup table, LUT)

spécifique à la requête, comme nous le verrons dans la section 2.2. La recherche par ADC est appelée *recherche asymétrique*, par opposition à la recherche dite *symétrique*, i.e., lorsque la requête est compressée ou quantifiée comme c'est le cas pour les vecteurs de la base de données. Par rapport au hachage, les approches par VQ offrent une plus grande précision, ce qui en fait une alternative largement utilisée (bien que le hachage soit plus rapide que la VQ).

Dans notre travail présenté dans le chapitre 3, nous étendons l'idée de la VQ en approchant les vecteurs de la base de données avec un *codage parcimonieux contraint*. Pour être plus précis, nous utilisons une somme pondérée de codewords pour représenter un vecteur de caractéristiques. Les *poids* de ces combinaison appartienent à un sous-ensemble fini de possibilités. En particulier, nous montrons comment nous pouvons reformuler deux quantificateurs vectoriels populaires, la quantification produit (*Product Quantization*, PQ) et la quantification vectorielle résiduelle (RVQ), à l'aide de notre approche de codage *parcimonieux contraint*, afin d'obtenir une meilleure reconstruction et une meilleure précision de recherche.

Apprentissage supervisé d'une représentation compacte

L'*apprentissage supervisé* a bénéficié à diverses tâches visuelles. L'apprentissage supervisé peut être défini comme l'apprentissage d'une fonction qui relie une entrée à une sortie, cohérente avec les paires entrée-sortie de l'ensemble d'apprentissage. L'apprentissage supervisé des réseaux profonds a permis d'atteindre des gains de performances remarquables pour de nombreuses tâches d'analyse visuelle telles que la classification, la localisation ou la recherche par le contenu, pour n'en citer que quelques-unes.

L'apprentissage supervisé par de CNN pour la classification d'images est largement utilisé pour l'extraction de représentations d'images à des fins variées. Ces représentations sont ensuite utilisées telles quelles ou après raffinement (généralement supervisé) pour résoudre différents problèmes tels que l'appariement d'images, la recherche par le contenu, la classification, le suivi, la localisation, etc. Ces représentations profondes se présentent sous la forme de vecteur de grande,voire très grande dimension. Bien que puissantes, elles ne sont donc pas adaptées à la recherche à grande échelle. Ceci est dû au fait que le calcul de distance ou de similarité entre des vecteurs de grande dimension est coûteux en temps calcul et nécessite également un stockage important. Aussi, il est important d'obtenir une représentation compressée. Certaines approches utilisent la VQ ou le hachage pour compresser la représentation, comme mentionné précédemment, tandis que d'autres approches sont fondées sur l'apprentissage supervisé de représentations compactes.

D'intéressants progrès ont récemment eu lieu dans le domaine de l'apprentissage de représentations compactes à l'aide CNN. La plupart des approches imposent à la représentation d'être binaire. Par analogie aux approches de hachage, celles-ci sont appelées méthodes de «hachage profond». La plupart de ces approches s'appuient sur un

entraînement supervisé pour améliorer les performances de recherche.

Sachant que les méthodes basées sur la VQ sont souvent plus performante que celles basées hashage, nous proposons une nouvelle méthode à base des réseaux de neurones convolutifs profonds produisant des codes binaires supervisés, compacts, *structurés*, pour la recherche visuelle. Cette nouvelle méthode peut être vue comme une variante de la VQ à base d'apprentissage profond, comme encodeur pour la recherche par le contenu. Notre méthode s'appuie sur une nouvelle *nonlinéarité softmax par blocs* et des nouvelles *fonctions de perte entropiques par batch* qui concourrent à une nouvelle structure dans les codes appris. Nous montrons que notre méthode surpasse celles *de l'état de l'art* basées sur le hachage profond ou la VQ pour la recherche de catégorie, la recherche d'instance et la classification. Ce travail est présenté en détail dans le chapitre 4.

Indexation inversée pour une recherche efficace

Jusqu'à présent, nous avons examiné plusieurs idées sur la façon d'obtenir une représentation compressée fonctionnelle et de chercher efficacement ces représentations à grande échelle. Il existe une autre approche complémentaire, qui est essentielle pour la recherche à très grande échelle, reposant sur une recherche non exhaustive.

La base de données de grande taille est divisée en plusieurs listes disjointes. Chacune de ces listes a un vecteur ou centroïde représentatif. Au moment de la recherche, la requête est comparée à tous ces centroïdes et seulement quelques listes sont sélectionnées. Seules les images de la base de données figurant dans ces listes sont ensuite comparées à la requête. Ainsi, la recherche est-elle limitée à un ensemble restreint de représentations, ce qui permet de limiter grandement le temps de recherche. Ces méthodes reposant sur la consultation uniquement d'un petit nombres de listes font appel à un "index inversé".

Dans le travail qui sera présenté dans cette thèse, nous étendons les idées présentées dans le chapitre 4 à l'apprentissage supervisé d'index inversés. En outre, nous proposons un apprentissage unifié des composants, de l'index inversé et de l'encodeur. Ce travail sur l'apprentissage supervisé de bout en bout d'un système complet d'indexation est présenté dans le chapitre 5.

Contenu et contributions de la thèse

Cette thèse porte sur la construction de systèmes de recherche par le contenu à grande échelle. La figure 1 montre le processus par lequel s'effectue la recherche, qui consiste en l'indexation de la base de données et la recherche des images les plus pertinentes étant donnée l'image requête. Il existe différents systèmes de recherche. La figure 2, met en exergue les particularités des différents systèmes de recherche existant, et des systèmes que nous avons conçus. Dans le chapitre 2, nous nous intéressons à plusieurs systèmes de recherche appliqués à la recherche à grande échelle, et au contexte dans lequel ils ont été conçus. Dans les trois chapitres qui suivent, nous présentons nos contributions.

- Dans le chapitre 3, nous traitons le problème de la recherche approximative du plus proche voisin. Nous proposons d'approcher les vecteurs de la base de données par un codage parcimonieux contraint, avec les poids atomiques (ou codeword normalisés) limités à un sous-ensemble fini. Contrairement aux méthodes de codage parcimonieux traditionnelles, le codage structuré quantifié proposé inclut l'utilisation de la mémoire comme contrainte de conception, ce qui nous permet d'indexer une collection importante telle que le benchmark BIGANN. Nos expériences, réalisées sur des benchmarks standards, montrent que notre formulation conduit à des solutions compétitives.
- Dans le chapitre 4, nous introduisons SUBIC, un code binaire structuré supervisé pour encoder des images pour une recherche efficace d'images à grande échelle. Le code binaire structuré est produit par un réseau neuronal convolutif profond supervisé en utilisant une non-linéarité *bloc-softmax* et des fonctions de perte entropique par batch. Nous montrons que notre méthode surpasse les représentations compactes de l'état de l'art basées sur le hachage profond ou la quantification structurée dans la recherche de catégorie intra et interdomaine, la recherche d'instance et la classification.
- Dans le chapitre 5, nous étendons le travail présenté au chapitre 4 pour *l'apprentissage de l'index complet*. Dans ce travail, nous proposons un premier système qui apprend les deux composants, un index inversé et un encodeur, dans un cadre neuronal unificateur de codage binaire structuré. Nous montrons que notre méthode atteint des résultats de pointe dans la recherche d'image à grande échelle.

Enfin, dans le chapitre 6, nous résumons nos contributions et discutons de quelques directions possibles.

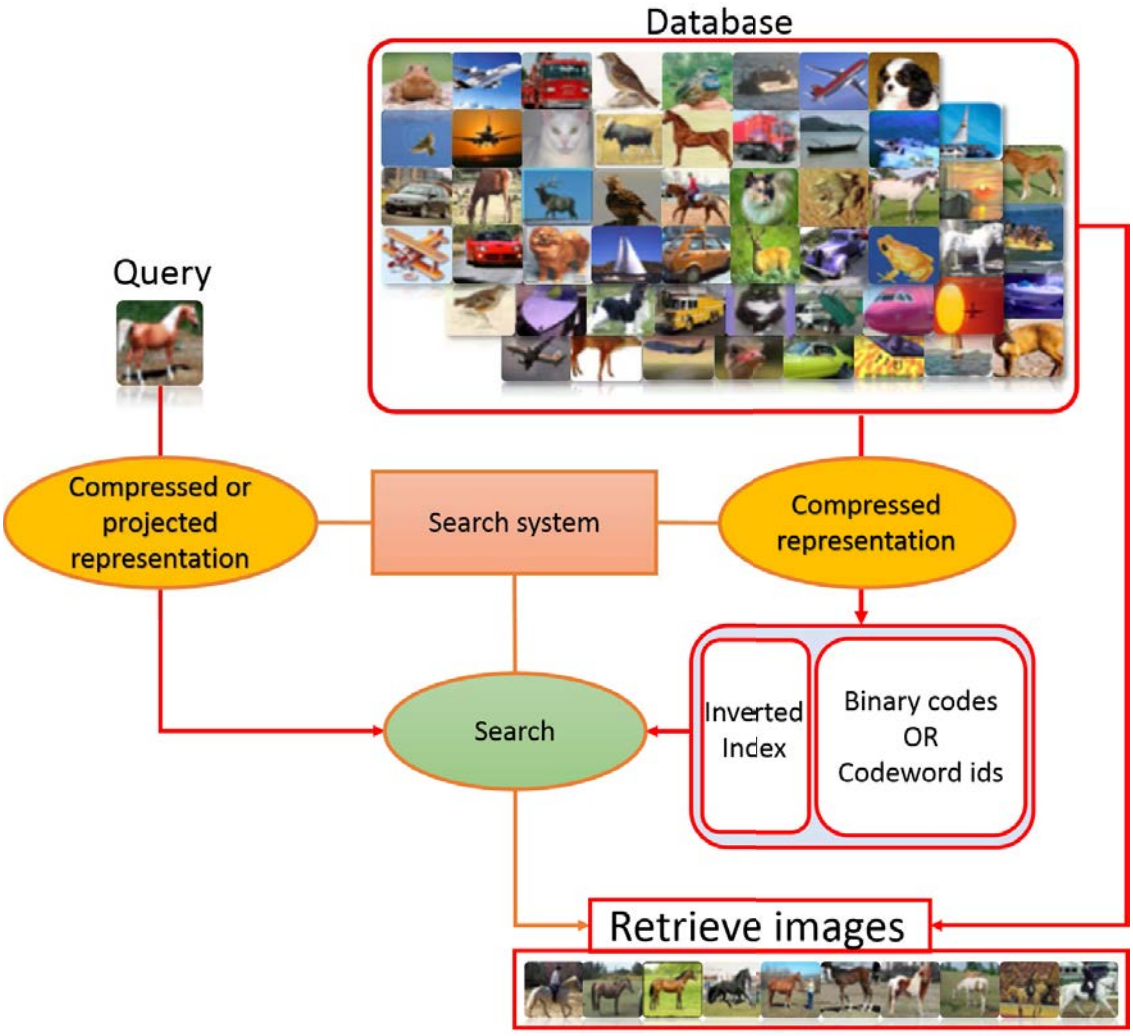


Figure 1: Pipeline de recherche d'images à grande échelle. Les images de la base de données sont indexées avec un *index inversé* et compressées avec des *codes compacts*. Pour la requête, soit elle est compressée (recherche symétrique), soit elle est projetée pour définir une table de correspondance (recherche asymétrique).

Search system	Compressed representation		Search
	Database	Query	
Hashing (Chapter 2)	Feature extraction Features encoded as Binary codes	Feature extraction Features encoded as Binary codes	symmetric search with hamming distance
Vector quantization (Chapters 2 and 3)	Feature extraction Features encoded as codeword ids	Feature extraction uncompressed	asymmetric search using look-up table
Deep Hashing (Chapter 2)	Images encoded as binary code	Images encoded as binary code	symmetric search with hamming distance
SUBIC (Chapter 4)	Images encoded as codeword ids	Uncompressed representation	asymmetric search

Inverted index	
Vector quantization based (Chapters 2 and 5)	Unsupervised clustering for inverted indexing. Sequential learning with encoder.
SUBIC based (Chapter 5)	Supervised clustering for inverted indexing. Possibility of joint learning with encoder.

Figure 2: Parties du système de recherche. Le tableau présente divers systèmes de recherche et nos contributions.

ABSTRACT

This thesis addresses the problem of large-scale image search. To tackle image search at large scale, it is required to encode images with *compact representations* which can be efficiently employed to compare images meaningfully. Obtaining such compact representation can be done either by compressing effective high dimensional representations or by learning compact representations in an end-to-end manner. The work in this thesis explores and advances in both of these directions.

In our first contribution, we extend *structured vector quantization* approaches such as Product Quantization by proposing a weighted codeword sum representation. We test and verify the benefits of our approach for approximate nearest neighbor search on local and global image features which is an important way to approach large scale image search.

Learning compact representation for image search recently got a lot of attention with various deep hashing based approaches being proposed. In such approaches, deep convolutional neural networks are learned to encode images into compact binary codes. In this thesis we propose a deep supervised learning approach for structured binary representation which is a reminiscent of structured vector quantization approaches such as PQ. Our approach benefits from asymmetric search over deep hashing approaches and gives a clear improvement for search accuracy at the same bit-rate.

Inverted index is another important part of large scale search system apart from the compact representation. To this end, we extend our ideas for supervised compact representation learning for building inverted indexes. In this work we approach inverted indexing with supervised deep learning and make an attempt to unify the learning of inverted index and compact representation.

We thoroughly evaluate all the proposed methods on various publicly available datasets. Our methods either outperform, or are competitive with the state-of-the-art.

CHAPTER

ONE

INTRODUCTION

1.1 Motivation and Problem

With the increased accessibility to Internet through connected devices, large amount of data is being generated and made available online. There are over 1.8 billion websites[1] on the world wide web each containing data in the form of text, images, audio files, and videos. This large amount of data brings opportunities and challenges to understand and analyze it or to search in it. To find documents or information of interest in this tremendous amount of data requires a powerful and efficient retrieval system. For such a retrieval system, it is essential to have some sense of understanding of the content and an efficient way to compare based on that understanding. These problems become even more challenging when the data is visual, as defining the semantics or content of the visual data is often subjective. And thus, finding a representation which can handle the possible different semantic meanings of the data is difficult. In our work, we try to address this challenging problem with various contributions towards an efficient *image retrieval system*.

An image retrieval system is supposed to index and encode the large database of images in such a way that, when a user query, the system should be able to retrieve the relevant images in real time. As we will see in the following sections, such a system relies on accurate and compact representation of the data.

This thesis work focuses on learning compact image representations and employing them for efficient and accurate image retrieval. In particular, it addresses the problem of large scale image search.

Large scale image search

The images are ubiquitous over the Internet and growing at a tremendous rate. For example, 147 thousand images are uploaded every minute just on Facebook[2]. So with this

[1] http://www.internetlivestats.com/total-number-of-websites/
[2] https://sproutsocial.com/insights/facebook-stats-for-marketers/

ever increasing visual data, thanks to the presence of camera in almost everyone's hand all the time, we need systems which allow us to search or rank images of interest. Further the search should be based on the visual content of the images which may or may not benefit from meta-data. For an accurate comparison of images, we need an image representation which is well suited for the kind of comparison we want. There are large research efforts on learning such representations from hand-crafted [114, 87, 102, 67] to convolutional neural network (CNN) [74, 116, 52, 47] based. The current state-of-the-art image representations are CNN-based and give excellent accuracy for identifying the visual content in the image (classification) or comparing images (similarity search/retrieval). These representations are high-dimensional typically ranging from 128 to 4096.

When the search is on a large scale, it has to be efficient. Efficiency comes from faster comparison of images and avoiding expensive disk operations by having all the image representative vectors on a faster memory like RAM.

Thus for large scale search, it is important to have compressed representation of the images as this allows the system to have all of the database in the main memory and avoids time consuming disk accesses. To see how compression helps let's consider an example of image search on a laptop. Suppose that we have 8GB of RAM available, it can fit only half a million feature vectors of 4096 dimension each of which needs 16KB (assuming 4-byte floating points) to store. While with compressed representation of 8B we can store a billion, i.e. 2000 times more. The billion images might be stored on a cloud server, all we need to do the search is the compressed representation of those images on our laptop. Once we find out the images of interest using the search system on compressed representation, we can retrieve them from the cloud storage.

Another essential requirement is of efficient and accurate computation of similarity or distance between the query and the database points. There has been a large research effort in learning image representations or *image features* which enable us to compare two images. Some of these approaches also constraint the features to be compact while there are other methods which compress high dimensional features to target large scale search.

In the following three sections, we will briefly discuss various ideas applicable for large scale search along with our contribution in those directions.

1.2 Finding the nearest neighbors

Suppose that we have an excellent feature extractor for images, say $f(.)$, and for an image I_i it gives a D dimensional vector $x_i = f(I_i) \in \mathbb{R}^D$. Further assume that, the similarity of images holds in the feature space in the form of distance between their features. Explicitly, if $d(x_i, x_j) < d(x_i, x_k)$ implies I_i and I_j are more similar than I_i and I_k, where $d(.,.)$

is some distance metric like the Euclidean distance. Now the problem of finding similar images becomes a problem of nearest neighbors search in the feature space. As a particular consequence of *curse of dimensionality*, for high dimensional features (which is mostly the case with images), any exact nearest neighbor search cannot have a search complexity better than the order of database size times D. Therefore on the large scale, exact nearest neighbor search needs prohibitively long time and also, requires large storage. As mentioned before it is important to have a compressed representation with an efficient way to compute the distance or similarity between the features. So with the practical constraints, it is not feasible to do exact distance computation but there are approaches which satisfy this practical constraints with an approximate distance computation. With this approximation we get approximate nearest neighbors (ANNs) which are the nearest neighbors with high probability, but not one.

The two most popular approaches to do ANN search are vector quantization (VQ) [50, 66] and hashing [57, 42].

In hashing methods for ANN search, the features are first mapped to a binary code using multiple hash functions. These mapping are supposed to approximately preserve the distance relationship between the features. And, the comparison is done with Hamming distance between the binary codes which is fast to do. In [19], it is shown that under some assumption, the Hamming distance is related to cosine similarity between original features.

The other approach is vector quantization for ANN search. It is a popular way to compress vectors. In VQ, the feature vectors are approximately represented by one or a combination of some other vectors called the *codewords*. This approximate representation leads to the compressed form, as each feature is represented by one or a few ids identifying the codeword(s). Now to find the nearest neighbors of a query, it is only compared to the approximate representations of the feature vectors which makes it very efficient. This comparison can be done even without quantizing the query, it is called *asymmetric distance computation* (ADC) [66]. Compared to when a quantized query is used, ADC leads to better distance approximation with the same complexity, thanks to the use of query specific *look-up table*, as we will see in Section 2.2. The search with ADC is referred to as *asymmetric search* while when the query is compressed or quantized as done for the database vectors, it is called *symmetric search*. In comparison to hashing, VQ approaches give much higher accuracy [66] making it a widely used alternative though hashing is faster than VQ.

In our work presented in Chapter 3 we extend the idea of VQ by approximating the database vectors with *constrained sparse coding*. To be more precise we use weighted sum of codewords to represent a feature vector. These codeword weights are restricted to belong to a finite subset. In particular we show how we can reformulate two popular vector quantizers like product quantization (PQ) [66] and residual vector quantization (RVQ)

[24], with our constrained sparse coding approach, to achieve better reconstruction and search accuracy.

1.3 Supervised learning of compact representation

Supervised learning has benefited various visual tasks. Supervised learning can be defined as the task of learning a function that maps an input to an output which is consistent with the input-output pairs of the training set. Supervised learning of deep networks has given remarkable accuracy on many visual tasks like classification, localization, retrieval to name a few.

Supervised learning with convolutional neural networks for image classification is widely used for image representation learning. These image features are then used as it is or with some tuning to address various problems like image matching, retrieval, classification, tracking, localization etc. These CNN based representations are usually high dimensional vectors, though highly accurate but is not efficient for large-scale search. This is due to the fact that the distance or similarity computation between high dimensional features has large time complexity and, they also require larger storage. Thus, it is important to get compressed representation. Some approaches employ VQ or hashing to compress the representation as mentioned before, while another idea is supervised learning of the compact representation.

Learning compact CNN based representation has recently made interesting progress. Most of the approaches are based on constraining the representation to be binary, thus, similar to the hashing approaches and hence these are called "Deep hashing" methods. Most of these approaches benefit from supervised training to give improved retrieval performance. Deep hashing is a deep learning variant of hashing approaches.

As we know that VQ based methods can give better accuracy, we propose a novel method to make deep convolutional neural networks produce supervised, compact, *structured binary codes* for visual search. This is like a deep learning variant of VQ like encoder for search. Our method makes use of a novel *block-softmax nonlinearity* and *batch-based entropy losses* that together induce structure in the learned encodings. We show that our method outperforms state-of-the-art compact representations based on deep hashing or VQ for category retrieval, instance retrieval and classification. This work is presented in detail in Chapter 4.

1.4 Inverted indexing for efficient search

Till now we have looked into the ideas about how to get a functional compressed representation and search efficiently using them to respect the practical constraints associated

with the large-scale search. There is another complementary approach which is essential for very large-scale search, that is to do *non-exhaustive search*.

The large collection or database of image representations is divided into multiple non-overlapping lists. Each of these lists has a representative vector or centroid. Now at search time, the query is compared to all these centroids and then only a few lists are selected. The database images which are in those lists are compared to the query and the rest of database is not considered. So basically, we limit our search to only a selected small subset of the database to highly reduce the search time. This non-exhaustive search system is first proposed in [66], where division and indexing of database points are done with K-means clustering, it is called *inverted file indexing* (IVF). While the comparison between the query and subset of database points is done with the help of product quantization.

In our work, we extend the ideas of our work [62] (presented in Chapter 4) for supervised learning of inverted index. Further we propose unified learning of both the parts, inverted index and encoder. This work on *learning the complete indexing pipeline* is presented in Chapter 5.

1.5 Thesis outline and contributions

This thesis focuses on building large scale search system. Figure 1.1 shows the search pipeline, which constitutes indexing the database and image retrieval based on the query image. There are various search systems as outlined in Figure 1.2, which highlights the peculiarities of different search systems along with ours. In Chapter 2 we discuss the background and some of these search systems for large scale search. Then, in the following three chapters we describe our contributions.

- In Chapter 3, we propose an extension of VQ approaches using *constrained sparse coding*. In this chapter we address the problem of approximate nearest neighbor search. Our formulation leads to competitive results on standard benchmarks for ANN search.
- In Chapter 4, we present our proposed supervised and end-to-end compact representation learning approach. In this work we introduce SUBIC, a *supervised structured binary codes* for efficient visual search.
- In Chapter 5, we extend the work presented in Chapter 4 for *learning a complete image indexing pipeline*. In this work, we propose a first system that learns both components, an inverted index and an encoder, within a unifying neural framework of structured binary encoding.

Finally, in Chapter 6, we summarize our contributions and discuss some possible future directions.

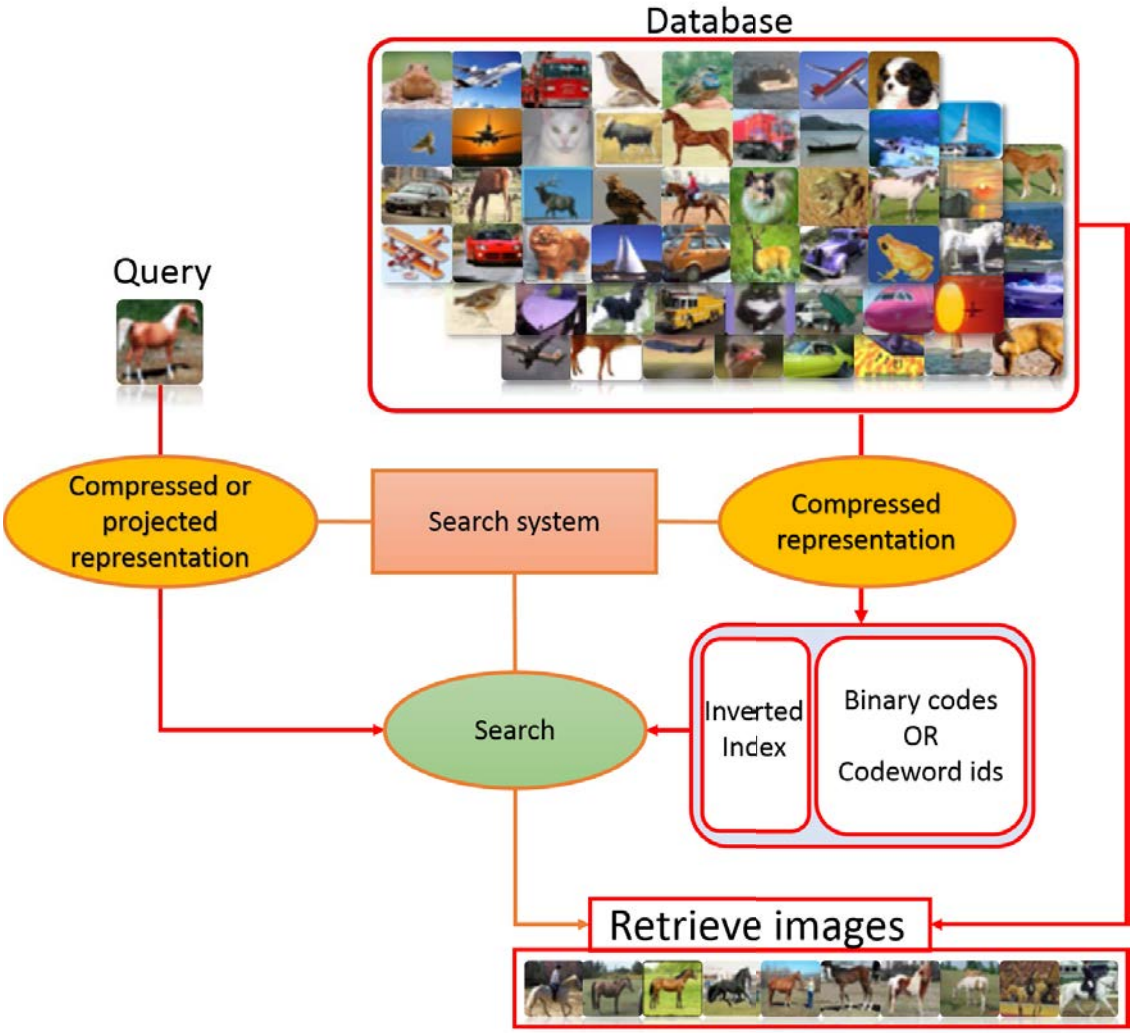


Figure 1.1: Large scale image search pipeline. The database images are indexed with *inverted index* and compressed with *compact codes*. For the query, either it is compressed (symmetric search) or it is projected to make a look-up table (asymmetric search).

Search system	Compressed representation		Search
	Database	Query	
Hashing (Chapter 2)	Feature extraction Features encoded as Binary codes	Feature extraction Features encoded as Binary codes	symmetric search with hamming distance
Vector quantization (Chapters 2 and 3)	Feature extraction Features encoded as codeword ids	Feature extraction uncompressed	asymmetric search using look-up table
Deep Hashing (Chapter 2)	Images encoded as binary code	Images encoded as binary code	symmetric search with hamming distance
SUBIC (Chapter 4)	Images encoded as codeword ids	Uncompressed representation	asymmetric search

Inverted index	
Vector quantization based (Chapters 2 and 5)	Unsupervised clustering for inverted indexing. Sequential learning with encoder.
SUBIC based (Chapter 5)	Supervised clustering for inverted indexing. Possibility of joint learning with encoder.

Figure 1.2: Parts of the search system. The table outlines various search systems and our contributions.

CHAPTER

TWO

BACKGROUND AND RELATED WORK

In this chapter, we build the background needed for the later chapters and review some of the relevant works. Large scale search and vector quantization methods are well studied and have very large literature. Thus it cannot be completely reviewed here. Therefore, we focus on the approaches which are more recent and relevant to our work. We will start with vector quantization approaches and their application in approximate nearest neighbor (ANN) search, which is central to our work. Then, we will briefly review hashing methods for ANN search and supervised deep learning methods for compact representation learning. Then in the last part of the chapter we give a short review of non-exhaustive search approaches.

2.1 Vector Quantization

Vector quantization (VQ) is very well studied in information theory [81, 72, 50, 51]. VQ is used for data compression to achieve speedy transmission or low storage requirement. The objective is to get a compressed representation with minimal loss of information or *reconstruction error*. It is usually formulated as an optimization problem to minimize the *mean squared error* (MSE) between the training data $\mathcal{Z}$ and its reconstruction,

$$\sum_{\mathbf{x}\in\mathcal{Z}} \|\mathbf{x} - Q(\mathbf{x}, \mathcal{C})\|_2^2. \tag{2.1}$$

The quantizer, $Q(\mathbf{x}, \mathcal{C})$ maps a data vector $\mathbf{x}$ to another vector from the set of vectors $\mathcal{C}$ called codebook. Sometimes we drop the explicit specification of the codebook $\mathcal{C}$ in that case we write the quantizer function as $Q(\mathbf{x})$.

Let the vectors in the codebook be $\mathbf{c}_k$, where $k \in \{1, \dots, |\mathcal{C}|\}$. The vectors $\mathbf{c}_j$ are called the codewords or the reproduction vectors. Also let the quantizer function for a given codebook $\mathcal{C}$ be defined as

$$Q(x, \mathcal{C}) = \mathbf{c}_{k(\mathbf{x})} \tag{2.2}$$

where $k(\mathbf{x})$ is the assignment index or codeword id given by,

$$k(\mathbf{x}) = \underset{k}{\operatorname{argmin}}\, d(\mathbf{x}, \mathbf{c}_k) \tag{2.3}$$

and the distance function $d(\mathbf{x}, \mathbf{y})$ is usually the Euclidean distance *i.e.* $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$ (which is apt for MSE minimization, but in some case such as normalized codewords, other distance functions can also be useful). Sometimes similarity is used instead of distance, in that case, $k(\mathbf{x})$ is found by argmax of the similarity between the data point and the codewords.

The codebook is learned to minimize the MSE over the training data $\mathcal{Z}$,

$$\min_{\mathcal{C}} \sum_{\mathbf{x} \in \mathcal{Z}} \|\mathbf{x} - Q(\mathbf{x}, \mathcal{C})\|_2^2 \tag{2.4}$$

Assuming that the training data is a good representative of the data to be encoded, the quantization should have low reconstruction error with the learned codebook.

In very popular and simple *K-means quantization* the codebook is learned with K-means algorithm for clustering. In the learning stage, the codebook is learned by alternating between the data point assignment as in Eq. (2.3) and codebook update with the mean of the new clusters, which is the closed form solution of Eq. (2.4). Now in the encoding stage, the database vectors are encoded by the codeword id based on the nearest codeword again as in the assignment equation Eq. (2.3). This gives the compressed representation of the data points as they are now stored by a codeword id or an integer rather than a high dimensional floating point vector. As the codeword id can range from 1 to K, where K is the codebook size $|\mathcal{C}|$ *i.e.* the number of codewords, it only needs $log_2(K)$ bits to store per data point. The data point is reconstructed by the codeword corresponding to the id.

To achieve a better reconstruction or lower *reconstruction error*, a large codebook is required. But larger K needs even larger training data, large storage for the training data and the codewords, and increased time complexity for learning the codebook. The time and storage complexity for the learning and encoding stage increases linearly with K though the memory requirement for compressed representation increases only logarithmically.

To this end, there are many vector quantization approaches which use multiple codebooks to achieve sub-linear storage and time complexity for learning and encoding. These approaches are distinct in the way they learn the codebooks or the constraints on the codebooks. In the following Sections, we describe some of such popular vector quantization approaches which use multiple codebooks.

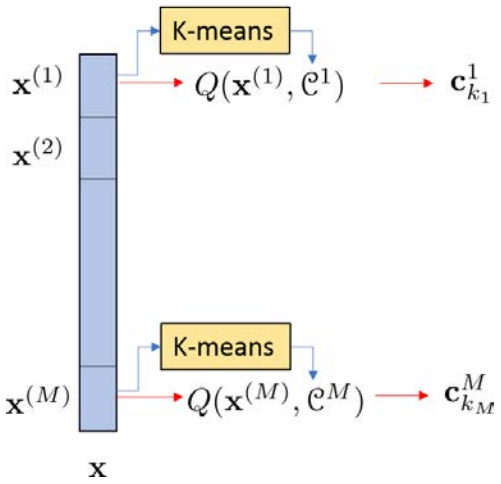


Figure 2.1: Product quantization. In PQ the vector is split, and a separate codebook is learned for each split (*learning stage*, in blue arrows). While encoding, each sub-vector from split is quantized using the corresponding codebook (*encoding stage*, in red arrows).

2.1.1 Product Quantization

Product quantization (PQ) is another simple vector quantization approach [66]. The data space is divided into M orthogonal subspaces simply by splitting each data vector into M sub-vectors as shown in Figure 2.1. More specifically, the data vector $\mathbf{x} \in \mathbb{R}^D$ is split into $\mathbf{x}^{(i)}$ for $i = \{1, \dots, M\}$ such that $\mathbf{x}$ is a concatenation of these sub-vectors *i.e.* $\mathbf{x} = [\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(M)}]$. For simplicity, consider D is divisible by M and each sub-vector $\mathbf{x}^{(i)} \in \mathbb{R}^{D/M}$. A codebook is learned for each sub-space on the sub-vectors from the training data. Let $\mathcal{C}^i$ be the codebook for i^{th} sub-space. As the sub-spaces are orthogonal, these codebooks are orthogonal to each other.

A data point is represented by concatenation of M codewords one from each codebook. Thus the effective set of representations or codewords is the Cartesian product of the codebooks *i.e.* $\mathcal{C} = \mathcal{C}^1 \times \dots \times \mathcal{C}^M$. The i^{th} codebook $\mathcal{C}^i$ is learned using K-means clustering on the i^{th} sub-vectors. Given that the codebooks are orthogonal

$$\mathbf{c}^{i\top}_{k_i} \mathbf{c}^j_{k_j} = 0, i \neq j \;\&\; i, j \in \{1, \dots, M\}, \tag{2.5}$$

we can rewrite the MSE minimization expression in Eq. (2.4) as,

$$\sum_{i=1}^{M} \min_{\mathcal{C}^i} \sum_{\mathbf{x} \in \mathcal{Z}} \|\mathbf{x}^{(i)} - Q(\mathbf{x}, \mathcal{C}^i)\|_2^2. \tag{2.6}$$

Note that the minimization over each codebook is independent of other codebooks as the codebooks are orthogonal.

With product codebooks, PQ achieves much higher representation capacity using very few codewords. For example, consider PQ with $M = 8$ codebooks and let each codebook have $K = 256$ codewords. There are only $M \times K$ codewords to learn and store while, due

to Cartesian product of the codebooks, there is a representation capacity of $K^M = 2^{64}$ codewords. It requires $M \times \log_2(K)$ bits to encode a data vector, in our example it is 64 bits.

By giving access to such a high representation capacity, PQ provides a very practical approach for better compression (low reconstruction error) and search which we will discuss in Section 2.2. There are various approaches which extend the idea of PQ, which we see next.

2.1.2 Optimized Product Quantization

In two concurrent works *Cartesian K-means* (CKM) [99] and *optimized product quantization* (OPQ) [39], it is pointed that the natural split of coordinates in PQ is not optimal. They formulate PQ as an optimization problem that minimizes reconstruction error by not just finding optimal codewords (as in PQ) but also optimal space decomposition. Their objective is to minimize,

$$\min_{\mathbf{R},\mathcal{C}} \sum_{\mathbf{x}\in\mathcal{Z}} \|\mathbf{R}\mathbf{x} - Q(\mathbf{R}\mathbf{x}, \mathcal{C})\|_2^2 \tag{2.7}$$

where $\mathbf{R} \in \mathbb{R}^{D\times D}$ is a rotation matrix and $\mathbf{R}^T\mathbf{R} = \mathbf{I}$.

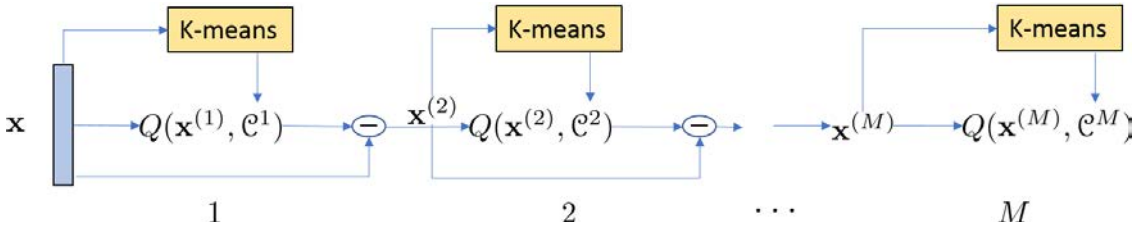
To learn $\mathbf{R}$ and $\mathcal{C}$, they alternatively optimize for one with the other fixed. When $\mathcal{C}$ is kept fixed, the optimization over $\mathbf{R}$ is an orthogonal Procrustes problem which has a closed-form solution [110]. When $\mathbf{R}$ is fixed it is similar to PQ, *i.e.* to find the optimal codebook with natural split given the *rotated* data.

2.1.3 Generalization with Additive Quantization and its variants

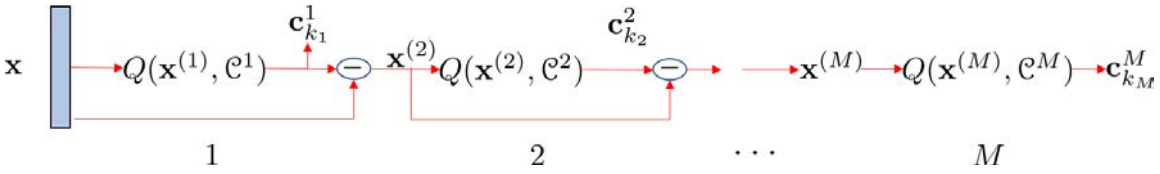
Additive quantization (AQ) [5] generalizes PQ by using unconstrained multiple full dimensional codebooks. In AQ, a data vector is represented by sum of codewords one from each of M codebooks. AQ attempts to minimize MSE in Eq. (2.4) with no constraint on the codebook such as orthogonality or spanning only a sub-space. This makes the encoding *i.e.*, finding the best set of codewords to represent a vector, very challenging. The encoding in AQ is a combinatorial problem which is NP-hard. In [5], they propose an approximate solution to this combinatorial problem, which achieves excellent reconstruction but at a very high time complexity making it less scalable.

Composite Quantization (CQ) [135] is similar to AQ. It uses multiple full dimensional non-orthogonal codebooks but with a constraint that the sum of the inner-products of all pairs of codewords (from different codebooks) that are used to represent any vector is constant *i.e.*,

$$\sum_{m=1}^{M}\sum_{m'=1}^{M} \mathbf{c}_{k_m(\mathbf{x})}^T \mathbf{c}_{k'_m(\mathbf{x})} = \epsilon. \tag{2.8}$$



(a) Learning stage



(b) Encoding stage

Figure 2.2: Residual vector quantization. RVQ sequentially quantizes the residuals using the codebook learned on the corresponding level of residuals.

This aides in faster search as we will see in the next Section 2.2.

Sparse Composite Quantization (SCQ) [136] further extends CQ by constraining the codewords to be sparse through minimizing ℓ_1-norm of the codewords, again this is also to make the search faster.

Another recent approach, *Local search quantization* (LSQ) [94] attempts to improve the encoding complexity of AQ based on stochastic local search. These methods partially improve the encoding complexity of AQ but overall still have a very demanding encoding stage compared to PQ or OPQ. See [96] for a good overview of PQ based approaches.

2.1.4 Residual Vector Quantization (RVQ)

RVQ is an old multi-codebook VQ approach [72][11]. The idea is to quantize a vector using multiple K-means quantizations, RVQ sequentially quantizes a vector and its *residuals* (reconstruction error). As shown in Fig. 2.2, a vector $\mathbf{x}^{(1)} = \mathbf{x}$ is first quantized using K-means quantization then, its residual $\mathbf{x}^{(2)} = \mathbf{x}^{(1)} - Q(\mathbf{x}^{(1)}, \mathcal{C}^1)$ is quantized and so on. The codebooks $\mathcal{C}^m$ is also learned sequentially using K-means clustering algorithm on the residuals $\mathbf{x}^{(m)}$ in sequence $m = 1$ to M, where

$$\mathbf{x}^{(m)} = \mathbf{x}^{(1)} - \sum_{i=1}^{m-1} Q(\mathbf{x}^{(i)}, \mathcal{C}^i) \tag{2.9}$$

and $\mathbf{x}^{(1)} = \mathbf{x}$.

A vector is represented as,

$$\mathbf{x} \approx \sum_{m=1}^{M} Q(\mathbf{x}^{(m)}, \mathcal{C}^m). \tag{2.10}$$

There is an extension of RVQ called *Enhanced RVQ* (ERVQ)[1]. It builds on RVQ by first learning the codebooks sequentially as in RVQ and then, iteratively optimizing each codebook keeping the others fixed for lower reconstruction error.

2.2 Vector quantization for ANN search

Finding the nearest neighbor of a vector in a collection of vectors has many important applications including image search. For a given query vector $\mathbf{y} \in \mathbb{R}^D$ and a database of vectors $\mathcal{X}$, the nearest neighbor (NN) search problem is to find a vector $\mathbf{x}^* \in \mathcal{X}$ such that

$$d(\mathbf{x}^*, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{y}), \forall \mathbf{x} \in \mathcal{X}$$

for some distance metric $d(.,.)$.

When the search is on a large scale with high dimensional vectors, finding the exact nearest neighbor is not practical. Therefore, we must rely on approximate nearest neighbor (ANN) search. The ANN search methods find the neighbors by efficiently computing approximate distance between vectors.

Vector quantization has been used for efficient and accurate ANN search. It was first proposed in [65, 66] which used PQ and then in [24] RVQ is used for ANN search. These works were followed by many other PQ-based [99, 39, 5, 40, 1, 136] and RVQ-based[1] methods.

In the previous section we have discussed VQ's use for compression with approximate representation. Now for an efficient search, the database is quantized, which compresses it. This is done offline. When searching for NN of a query $\mathbf{y}$, it is compared to the approximate representations of the database vectors to get approximation of the distances between them. There are two ways to compute the approximate distance based on whether the query is quantized or not.

Symmetric distance computation (SDC). The query is also quantized, and the quantized representations are compared for approximating the distance,

$$d(\mathbf{y}, \mathbf{x}) \approx d(Q(\mathbf{y}), Q(\mathbf{x})). \tag{2.11}$$

As $Q(.)$ is a codeword from a finite codebook, we can pre-compute all the pairwise distances between the codewords and store them in a *look-up table*. Thus we have approximate value of $d(Q(\mathbf{y}), Q(\mathbf{x}))$ for any $\mathbf{y}$ and $\mathbf{x}$. So at search time, $\mathbf{y}$ is quantized and the value of $d(Q(\mathbf{y}), Q(\mathbf{x}))$ for all $\mathbf{x} \in \mathcal{X}$ is fetched from the *look-up table*, leading to fast distance computation. In case of multiple codebooks, there will be a look-up table per codebook. We fetch and combine the values from each of them to get the approximate distance.

Asymmetric distance computation (ADC). The unquantized query is compared with the quantized database vectors

$$d(\mathbf{y}, \mathbf{x}) \approx d(\mathbf{y}, Q(\mathbf{x})). \tag{2.12}$$

When $d(,)$ is the squared Euclidean distance we can write Eq. (2.12) as,

$$d(\mathbf{y}, \mathbf{x}) \approx \|\mathbf{y} - Q(\mathbf{x})\|_2^2. \tag{2.13}$$

If the quantizer has multiple *orthogonal codebooks* like PQ, we can rewrite Eq. (2.13) as

$$d(\mathbf{y}, \mathbf{x}) \approx \sum_{m=1}^{M} \|\mathbf{y}^{(m)} - Q(\mathbf{x}, \mathcal{C}^m)\|_2^2 = \sum_{m=1}^{M} \|\mathbf{y}^{(m)} - \mathbf{c}_{k_m(\mathbf{x})}\|_2^2. \tag{2.14}$$

For a query $\mathbf{y}$ we can compute $\|\mathbf{y}^{(m)} - \mathbf{c}_{k_m(\mathbf{x})}\|_2^2$ for all the codewords for $m = \{1, 2, \dots, M\}$, and store these KM values in the look-up tables. Thus at search time, $d(\mathbf{y}, \mathbf{x})$ is approximated by a sum of M scalars which we get from the look-up table.

If the quantizer has multiple *non-orthogonal codebooks* like RVQ or AQ, Eq. (2.13) is not equivalent to Eq. (2.14), but can be rewritten as

$$d(\mathbf{y}, \mathbf{x}) \approx \|\mathbf{y}\|_2^2 + \|Q(\mathbf{x})\|_2^2 - 2 \sum_{m=1}^{M} \mathbf{y}^T \mathbf{c}_{k_m(\mathbf{x})}. \tag{2.15}$$

To rank the database vectors given the query $\mathbf{y}$, we do not need $\|\mathbf{y}\|_2^2$ as it doesn't affect the ranking. The computation of $\sum_{m=1}^{M} \mathbf{y}^T \mathbf{c}_{k_m(\mathbf{x})}$ benefits from the look-up table as before. To handle the squared norm of the database vector, $\|Q(\mathbf{x})\|_2^2$, there are two ways which can be used. First, as

$$\|Q(\mathbf{x})\|_2^2 = \|\sum_{m=1}^{M} \mathbf{c}_{k_m(\mathbf{x})}\|_2^2 = \sum_{m=1}^{M} \sum_{m'=1}^{M} \mathbf{c}_{k_m(\mathbf{x})}^T \mathbf{c}_{k'_m(\mathbf{x})}, \tag{2.16}$$

we can use a look-up table to store the dot-products between each pair of codewords and fetch them to compute the squared norm. This will require $\mathcal{O}(M^2)$ time complexity. Another way is that we compute the squared norm $\|Q(\mathbf{x})\|_2^2$ at the time of encoding, and store it with a single byte using a non-uniform scalar quantizer.

In composite quantization (CQ) [135], as mentioned in Section 2.1.3, the codebooks are non-orthogonal but are constrained to have nearly constant sum of product of inter-codebook codewords as in Eq. (2.8). Thus there is no need to compute the norm for the ANN search. The sparse composite quantization (SCQ) [136] imposes additional constraint on the CQ codebooks to be sparse, leading to efficient computation of the look-up table.

Comparison of ADC and SDC. As in ADC the query is not quantized, it usually gives better distance approximation compared to SDC. The search time complexity is same for both, as in ADC we build a look-up table by computing the distance (or similarity) between query and the codewords, and in SDC we do the same computation for quantizing the query. Thus ADC is mostly used for VQ based ANN search.

2.3 Binary Hashing for ANN search

Binary hashing is another very popular way to compress a database and search in it. In the binary hashing methods the data is mapped to a binary code. Consider a collection of data vectors, the objective of hashing methods is to encode them with binary codes such that the vectors which are close in the vector space have similar codes. To encode with a B-bit binary code, B binary hash function are used. Let $\mathcal{H} = \{h_1, h2, \dots, h_B\}$ be the collection of hash functions, the hashing method encodes a vector $\mathbf{x}$ as

$$h(\mathbf{x}) = \{h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_B(\mathbf{x})\} \tag{2.17}$$

where $h_j : \mathbb{R}^D \mapsto \{0, 1\}$ or $h_j : \mathbb{R}^D \mapsto \{-1, 1\}$. The function $h(.)$ maps the vector to a *Hamming space*.

To compare two data points or vectors say $\mathbf{y}$ and $\mathbf{x}$, the Hamming distance is used *i.e.*,

$$d_{\mathcal{H}}(\mathbf{y}, \mathbf{x}) = \sum_{j=1}^{B} |h_j(\mathbf{y}) - h_j(\mathbf{x})| = \sum_{j=1}^{B} h_j(\mathbf{y}) \oplus h_j(\mathbf{x}) \tag{2.18}$$

where $\oplus$ is logical XOR operation. This equation considers $h(\mathbf{x}) \in \{0, 1\}^B$, in case of $\{-1, 1\}^B$ it can be easily converted with shifting by one and dividing by two.

For ANN search, the database is encoded by the hash functions into binary hash codes thus compressing the database. At search time, the query is also encoded by the same hash functions and compared to the database vectors using Eq. (2.18) which is very efficient.

A commonly used hash function is defined as

$$h_j(\mathbf{x}) = sgn(f(\mathbf{w}_j^T \mathbf{x} + b_j)) \tag{2.19}$$

where sgn is a sign function which is 1 if $sgn(.) > 0$ and 0 or -1 otherwise. $f(.)$ is a prespecified function it could be a non-linear function or an identity mapping. $\mathbf{w}_j$ is a projection vector with intercept b_j.

Many hashing approaches are based on *Locality Sensitive Hashing* (LSH) [57, 42, 19, 29, 89, 49]. It is widely used for many applications [15, 16, 48, 93, 105]. The hash functions in LSH are supposed to assign the same bit to close vectors with high probability. Several LSH approaches use *Random Projection based Hashing* (RPH) *i.e.*, in the hash function, the vector $\mathbf{w}$ is a random projection vector and b is a random shift. Though these methods provide theoretical guarantees, they need large binary codes to get search precision which reduces the recall. Multiple hash tables can be used to overcome this problem [29] but then, it increases the storage and search time. Various strategies are proposed to overcome these issues, including LSH Forest [12], MultiProbe LSH [89][32] and others [109][28]. However, these random projection based hash functions are independent of

the data they are applied on, thus they miss the possibility to benefit from the specific properties or structure present in the data.

There are other approaches which use data-dependent hash functions. Many of these methods leverage machine learning to learn the hash functions. LSH has been combined with various machine learning techniques including kernel learning [75][97], metric learning [76] and others [70][98]. A recent work [33] proposed Polysemous codes which index the vectors using PQ with the codeword ids acting as the hash codes. At search time, the Hamming distance of the hash codes is used to filter out most of the vectors and then PQ is used to rank the remaining.

Among data-dependent hashing approaches there are several *unsupervised* and *supervised* approaches. The *unsupervised* hashing methods attempt to benefit from the distribution or structure of the data to learn effective hash codes, these methods include spectral hashing [127], graph hashing [85], manifold hashing [112], iterative quantization hashing [101], spherical hashing [53]. While *supervised* or *semi-supervised* hashing benefit from labeled data, some recent approaches include [100, 118, 111, 63]. See [124] for an overview of these binary hashing methods. Further, there are several recent supervised hashing based on deep learning, which we review next.

2.4 Deep Hashing

Deep learning has shown a great success in visual *representation learning* [46]. The learned deep representations have given breakthrough performance in various computer vision tasks [74, 126, 43, 117, 44, 116, 107, 86, 52, 35, 34]. These deep representations or deep features have also lead to improvements in image retrieval tasks [9, 2, 47]. Deep features along with ANN search approaches based on VQ or hashing can be and have been used successfully for large scale image search [23, 20, 8]. Still, in the recent years, there has been a lot of interest in constraining deep features to be binary and thus learning the deep network to do hashing. These deep hashing methods have demonstrated that the learned compact deep features improve retrieval performance compared to just applying ANN search approaches on top of the high dimensional deep features.

The deep hashing approaches employ a pre-trained or randomly initialized *base CNN* which produces high-dimensional feature vector for a given input image. This vector is then fed to another fully connected layer which is, at test time, followed by element-wise thresholding to give compact binary code, $\mathbf{h} \in \{0,1\}^B$ or $\mathbf{h} \in \{-1,1\}^B$. While at training time, usually *sigmoid* or *tanh* non-linearities replace thresholding to keep it differentiable and to allow gradient based training.

In case of supervised training on deep hashing networks, they can be grouped into *point-wise*, *pair-wise* and *triplet-wise* methods, based on the way they use the supervision. In *point-wise* training, the network is trained for image classification with *cross-entropy loss*.

In *pair-wise*, the network is trained with pairs of similar or dissimilar images with *contrastive loss* which makes the learned code (dis)similar for images which are (dis)similar. While in *triplet-wise*, the network trains on an image triplet (I_a, I_p, I_n) consisting an anchor image I_a, a similar image to the anchor I_p and a dissimilar image I_n. It uses *ranking loss*, the objective here is to have the similarity between the codes of similar images I_a and I_p, more than I_a and I_n. Below we look into a few recent supervised deep hashing approaches.

Deep semantic ranking based hashing (DSRH) [137] uses triplet-wise training with *ranking loss* and additionally, it penalizes the average of each bit over the training set such that code distribution is approximately zero-centered (the code is in $\{-1, 1\}^B$).

Deep regularized similarity comparison hashing (DRSCH) [134] also uses the triplet-wise supervised training and minimizes the ranking loss in Hamming space. It learns bit-wise weights along with the binary encoder, which results in richer codes but more costly distances to compute at search time.

In *Hashing with Deep Neural Network* (DNNH) [78] a piece-wise thresholding function which is sigmoid in the middle and, 0 and 1 on the edges to enforce the binarization and the network is trained for *ranking loss*.

Deep learning of binary hash codes (DLBHC) [80] is a simple approach with sigmoid activation for encoding and trained for image classification on the codes produced after sigmoid activations.

CNN hashing (CNNH) [129] proposes a two stage method, in the first stage it learns target binary codes using low rank factorization of a full pairwise similarity matrix. Then, in the second stage the target codes provide the supervision to train the CNN.

In *deep supervised hashing* (DSH) [82], a W-shaped loss with minima at the desired code values, to impose binarization in the code, is used along with *contrastive loss* for pair-wise training.

See the Table 4.1 for an overview of these approaches and its comparison to our proposed approach of *structured binary code*.

2.5 Non-exhaustive search

With compact codes and efficient ways to compare the query and database points, we can achieve a great speed up in the search and highly reduced memory requirements. But still it is an exhaustive search that is, we have to compare the query to all the database points. When the search scale is very large with tens of million to billion sized database, exhaustive search would be slow. Jégou *et al*. in [66] addressed this problem by building an inverted file system [114] to partition the database into mutually exclusive subsets

and at search time only a few of these subsets are searched for a given query.

To build the inverted index, a *coarse-quantizer* is used to partition the feature space into Voronoi cells. Corresponding to each cell, there is a list which contains the ids of all the database vectors that fall into it. Let the coarse codebook be $\mathcal{D} = \{\mathbf{d}_n \in \mathbb{R}^D\}_{n=1}^N$, a database vector $\mathbf{x}$ will be assigned to a list $\mathcal{L}_{n(x)}$, where $n(x)$ is found by solving,

$$n(\mathbf{x}) = \underset{k}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{d}_k\|_2. \tag{2.20}$$

This partitions the database into N list $\{\mathcal{L}_k\}_{k=1}^N$.

In [66], the inverted file (IVF) is used along with PQ for asymmetric distance computation (ADC) for ANN search. First the IVF is build using K-means as the *coarse-quantizer* and then, PQ is used to encode the residual

$$\mathbf{r}_{n(\mathbf{x})} = \mathbf{x} - \mathbf{d}_{n(\mathbf{x})}, \tag{2.21}$$

that is the difference between the database vector and the coarse quantizer's codeword.

At search time, the query vector is compared to all the codewords in $\mathcal{D}$ and based on the closest codewords, a few lists are selected to be searched. For each list, first the query's residual is computed as in Eq. (2.21) which is then used for building the look-up table for PQ codewords followed by ADC for all the points in the list. Note that, a look-up table has to be built for each list.

Babenko *et al*. in [4] proposed *Inverted Multi-Index* (IMI) which extend IVF by using product quantization as the coarse-quantizer. They used PQ with two codebooks, which produces $N = K^2$ inverted lists if each codebook has K codewords. As it partitions the database very densely, only a small number of candidates are good enough to give high recall. For a given query, to select the best lists of the candidates, [4] propose a *multi-sequence algorithm* that outputs the pairs of codewords (from different codebooks) ordered by the distance from the query. Then, the selected candidates are ranked using ADC based on the PQ encoder. In [59] a branch and bound algorithm is proposed to improve the original multi-sequence algorithm.

Some improvements on these inverted indexing approaches have been proposed recently. In [73], a separate OPQ is learned for each inverted list. In [6] improvements are done to IMI by using OPQ for the indexing and the encoding instead of PQ and also, learning a separate OPQ as in [73].

2.6 Conclusion

In this Chapter, we presented the approaches used for large scale search. We discussed about vector quantization and binary hashing for ANN search where the database vectors are compressed with codeword ids (VQ) or binary codes (hashing) and an efficient search mechanism is utilized for faster search. We also reviewed Deep hashing

approaches where CNNs are trained to produce compact binary codes. And, the last Section covered, non-exhaustive search with inverted indexing.

In the following Chapters, we propose various approaches, including constrained sparse coding based extension of VQ, supervised structured binary codes for encoding, and supervised learning of inverted indexing followed by joint learning of complete image indexing pipeline.

CHAPTER

THREE

APPROXIMATE SEARCH WITH QUANTIZED SPARSE REPRESENTATIONS

This chapter concerns approximate nearest neighbor search. We propose to approximate database vectors by constrained sparse coding, with the atom (or normalized codeword) weights restricted to a finite subset. This chapter is based on the following publication:

Approximate search with quantized sparse representations. H. Jain, P. Pérez, R. Gribonval, J. Zepeda, and H. Jégou. In proceedings of *European Conference on Computer Vision (ECCV)*, 2016.

Retrieving, from a very large database of high-dimensional vectors, the ones that "resemble" most a query vector is at the heart of most modern information retrieval systems. Online exploration of very large media repositories, for tasks ranging from copy detection to example-based search and recognition, routinely faces this challenging problem. Vectors of interest are abstract representations of the database documents that permit meaningful comparisons in terms of distance and similarity. Their dimension typically ranges from a few hundreds to tens of thousands. In visual search, these vectors are ad-hoc or learned descriptors that represent image fragments or whole images.

Searching efficiently among millions or billions of such high-dimensional vectors requires specific techniques. The classical approach is to re-encode all vectors in a way that allows the design of a compact index and the use of this index to perform fast *approximate search* for each new query. Among the different encoding approaches that have been developed for this purpose, state-of-the-art systems rely on various forms of vector quantization: database vectors are approximated using compact representations that can be stored and searched efficiently, while the query need not be approximated (asymmetric approximate search). In order to get high quality approximation with practical complexities, the encoding is structured, typically expressed as a sum of codewords stemming from suitable codebooks. There are two main classes of such structured quantization techniques: those based on vector partitioning and independent quantization of sub-vectors [39, 66, 99]; those based on sequential residual encoding [1, 24, 68, 135, 136].

In this chapter, we show how these approaches can be taken one step further by drawing inspiration from the sparse coding interpretation of these techniques [119]. The key idea is to represent input vectors as linear combinations of atoms, instead of sums of codewords. The introduction of scalar weights allows us to extend both residual-based and partitioned-based quantizations such that approximation quality is further improved with modest overhead. For this extension to be compatible with large scale approximate search, the newly introduced scalar weights must be themselves encoded in a compact way. We propose to do so by quantizing the vector they form. The resulting scheme will thus trade part of the original encoding budget for encoding coefficients. As we shall demonstrate on various datasets, the proposed quantized sparse representation (i) competes with partitioned quantization for equal memory footprint and lower learning/coding complexity and (ii) outperforms residual quantization with equal or smaller memory footprint and learning/coding complexity.

In the next section, we discuss in more details the problem of approximate vector search with structured quantizers and recall useful concepts from sparse coding. With these tools in hand, we introduce in Section 3.2 the proposed structured encoding by quantized sparse representations. The different bricks –learning, encoding and approximate search– are presented in Sections 3.3 and 3.4, both for the most general form of the framework (residual encoding with non-orthogonal dictionaries) and for its partitioned variant. Experiments are described and discussed in Section 3.5.

3.1 Related work

Approximate vector search is a long-standing research topic across a wide range of domains, from communication and data mining to computer graphics and signal processing, analysis and compression. As mentioned in Section 2.3, important tools have been developed around hashing techniques [57], which turn the original search problem into the one of comparing compact codes, *i.e.*, binary codes [19], see [124] for a recent overview on binary hashing techniques. Among other applications, visual search has been addressed by a number of such binary encoding schemes (*e.g.*, [88, 101, 118, 123, 130]).

An important aspect of hashing and related methods is that their efficiency comes at the price of comparing only codes and not vectors in the original input space. In this chapter we focus on another type of approaches that are currently state-of-art in large scale visual search. Sometimes referred to as *vector compression* techniques, they provide for each database vector $\mathbf{x}$ an approximation $Q(\mathbf{x}) \approx \mathbf{x}$ such that (i) the Euclidean distance (or other related similarity measure such as inner product or cosine) to any query vector $\mathbf{y}$ is well estimated using $Q(\mathbf{x})$ instead of $\mathbf{x}$ and (ii) these approximate (di)similarity measures can be efficiently computed using the code that defines $Q(\mathbf{x})$.

A simple way to do that is to rely on vector quantization [41], which maps $\mathbf{x}$ to the closest vector in a codebook learned through k-means clustering. In high dimensions though, the complexity of this approach grows to maintain fine grain quantization. As discussed in Section 2.1, a simple and powerful way to circumvent this problem is to partition vectors into smaller dimensional sub-vectors that are then vector quantized (see Figure 2.1). At the heart of product quantization (PQ) [66], this idea has proved very effective for approximate search within large collections of visual descriptors. Different extensions, such as "optimized product quantization" (OPQ) [39] and "Cartesian k-means" (CKM) [99] optimize the chosen partition, possibly after rotation, such that the distortion $\|\mathbf{x} - Q(\mathbf{x})\|$ is further reduced on average. Additionally, part of the encoding budget can be used to encode this distortion and improve the search among product-quantized vectors [54].

It turns out that this type of partitioned quantization is a special case of *structured* or layered quantization:

$$Q(\mathbf{x}) = \sum_{m=1}^{M} Q(\mathbf{x}, \mathcal{C}^m), \tag{3.1}$$

where $\mathcal{C}^m$ is a codebook. In PQ and its variants, these codebooks are orthogonal, which makes learning, encoding and search especially efficient. Sacrificing part of this efficiency by relaxing the orthogonality constraint can nonetheless provide better approximations. As discussed in Section 2.1, a number of recent works explore this path.

"Additive quantization" (AQ) [5] is probably the most general of those, hence the most complex to learn and use. It indeed addresses the combinatorial problem of jointly finding the best set of M codewords in (3.1). While excellent approximation and search performance is obtained, its high computational cost makes it less scalable [7]. In particular, it is not adapted to the very large scale experiments we report in this work.

In "composite quantization" (CQ) [135], the overhead caused at search time by the non-orthogonality of codebooks is alleviated by learning codebooks that ensure $\|Q(\mathbf{x})\| = \text{cst}$. This approach can be sped up by enforcing in addition the sparsity of codewords [136]. As AQ –though to a lesser extent– CQ and its sparse variant have high learning and encoding complexities.

A less complex way to handle sums of codewords from non-orthogonal codebooks is offered by the greedy approach of "residual vector quantization" (RVQ) [11, 72]. The encoding proceeds sequentially such that the m-th quantizer encodes the *residual* $\mathbf{x} - \sum_{n=1}^{m-1} Q(\mathbf{x}, \mathcal{C}^n)$. Accordingly, codebooks are also learned sequentially, each one based on the previous layer's residuals from the training set. This classic vector quantization approach was recently used for approximate search [1, 24, 95]. "Enhanced residual vector quantization" (ERVQ) [1] improves the performance by jointly refining the codebooks in a final training step, while keeping purely sequential the encoding process.

Important to the present work, *sparse coding* is another powerful way to approximate

and compress vectors [36]. In this framework, a vector is also approximated as in (3.1), but with each $Q(\mathbf{x}, \mathcal{C}^m)$ being of the form $\alpha_m \mathbf{c}_{k_m}$, where α_m is a scalar weight and $\mathbf{c}_{k_m}$ is a unit norm *atom* from a learned *dictionary*. The number of selected atoms can be predefined or not, and these atoms can stem from one or multiple dictionaries. A wealth of techniques exist to learn dictionaries and encode vectors [36, 91, 128], including ones that use the Cartesian product of sub-vector dictionaries [40] similarly to PQ or residual encodings [133, 132] similarly to RQ to reduce encoding complexity. Sparse coding thus offers representations that are related to structured quantization, and somewhat richer. Note however that these representations are not discrete in general, which makes them *a priori* ill-suited to indexing very large vector collections. Scalar quantization of the weights has nonetheless been proposed in the context of audio and image compression [133, 38, 131].

Our proposal is to import some of the ideas of sparse coding into the realm of approximate search. In particular, we propose to use sparse representations over possibly non-orthogonal dictionaries and with vector-quantized coefficients, which offer interesting extensions of both partitioned and residual quantizations.

3.2 Quantized sparse representations

A sparse coding view of structured quantization Given M codebooks, structured quantization represents each database vector $\mathbf{x}$ as a sum (3.1) of M codewords, one from each codebook. Using this decomposition, search can be expedited by working at the codeword level (see Section 3.4). Taking a sparse coding viewpoint, we propose a more general approach whereby M dictionaries[1], $\mathcal{C}^m = [\mathbf{c}_1^m \cdots \mathbf{c}_K^m]_{D \times K}$, $m = 1 \cdots M$, each with K normalized atoms, are learned and a database vector $\mathbf{x} \in \mathbb{R}^D$ is represented as a linear combination:

$$Q(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m(\mathbf{x}) \mathbf{c}_{k_m(\mathbf{x})}^m, \tag{3.2}$$

where $\alpha_m(\mathbf{x}) \in \mathbb{R}$ and $k_m(\mathbf{x}) \in [\![1, K]\!]$. Next, we shall drop the explicit dependence in $\mathbf{x}$ for notational convenience. As we shall see in Section 3.5 (Fig. 3.1), the additional degrees of freedom provided by the weights in (3.2) allow more accurate vector approximation. However, with no constraints on the weights, this representation is not discrete, spanning a union of M-dimensional sub-spaces in $\mathbb{R}^D$. To produce compact codes, it must be restricted. Before addressing this point, we show first how it is obtained and how it relates to existing coding schemes.

If dictionaries are given, trying to compute $Q(\mathbf{x})$ as the best ℓ_2-norm approximation of $\mathbf{x}$ is a special case of sparse coding, with the constraint of using exactly one atom from

[1]Throughout we use the terminology *codebook* for a collection of vectors, the *codewords*, that can be added, and *dictionary* for a collection of normalized vectors, the *atoms*, which can be linearly combined.

each dictionary. Unless dictionaries are mutually orthogonal, it is a combinatorial problem that can only be solved approximately. Greedy techniques such as projection pursuit [37] and matching pursuit [92] provide particularly simple ways to compute sparse representations. We propose the following pursuit for our problem: for $m = 1 \cdots M$,

$$k_m = \underset{k \in [\![1,K]\!]}{\operatorname{argmax}} \mathbf{r}_m^\top \mathbf{c}_k^m, \quad \alpha_m = \mathbf{r}_m^\top \mathbf{c}_{k_m}^m, \tag{3.3}$$

with $\mathbf{r}_1 = \mathbf{x}$ and $\mathbf{r}_{m+1} = \mathbf{r}_m - \alpha_m \mathbf{c}_{k_m}^m$. Encoding proceeds recursively, selecting in the current dictionary the atom with maximum inner-product with the current residual. Note that we use maximum and not *absolute maximum* inner-product as in matching pursuit. This permits to get a tighter distribution of weights, which will make easier their subsequent quantization. Once atoms have all been sequentially selected, *i.e.*, the support of the M-sparse representation is fixed, the approximation (3.2) is refined by jointly recomputing the weights as

$$\hat{\boldsymbol{\alpha}} = \underset{\boldsymbol{\alpha} \in \mathbb{R}^M}{\operatorname{argmin}} \|\mathbf{x} - C(\mathbf{k})\boldsymbol{\alpha}\|_2^2 = C(\mathbf{k})^\dagger \mathbf{x}, \tag{3.4}$$

with $\mathbf{k} = [k_m]_{m=1}^M \in [\![1, K]\!]^M$ the vector formed by the selected atom indices, $C(\mathbf{k}) = [\mathbf{c}_{k_1}^1 \cdots \mathbf{c}_{k_M}^M]_{D \times M}$ the corresponding atom matrix and $(\cdot)^\dagger$ the Moore-Penrose pseudo-inverse. Vector $\hat{\boldsymbol{\alpha}}$ contains the M weights, out of KM, associated to the selected support. Note that the proposed method is related to [133, 132].

Learning dictionaries In structured vector quantization, the M codebooks are learned on a limited training set, usually through k-means. In a similar fashion, k-SVD on a training set of vectors is a classic way to learn dictionaries for sparse coding [36]. In both cases, encoding of training vectors and optimal update of atoms/codewords alternate until a criterion is met, starting from a sensible initialization (*e.g.*, based on a random selection of training vectors). Staying closer to the spirit of vector quantization, we also rely on k-means in its spherical variant which fits well our needs: spherical k-means iteratively clusters vector *directions*, thus delivering meaningful unit atoms.

Given a set $\mathcal{Z} = \{\mathbf{z}_1 \cdots \mathbf{z}_R\}$ of R training vectors, the learning of one dictionary of K atoms proceeds iteratively according to:

$$\text{Assignment} : k_r = \underset{k \in [\![1,K]\!]}{\operatorname{argmax}} \mathbf{z}_r^\top \mathbf{c}_k, \ \forall r \in [\![1, R]\!], \tag{3.5}$$

$$\text{Update} : \mathbf{c}_k \propto \sum_{r:k_r=k} \mathbf{z}_r, \ \|\mathbf{c}_k\| = 1, \ \forall k \in [\![1, K]\!]. \tag{3.6}$$

This procedure is used to learn the M dictionaries. The first dictionary is learned on the training vector themselves, the following ones on corresponding residual vectors. However, in the particular case where dictionaries are chosen within prescribed mutually orthogonal sub-spaces, they can be learned independently after projection in each-subspace, as discussed in Section 3.3.

Quantizing coefficients To use the proposed representation for large-scale search, we need to limit the possible values of coefficients while maintaining good approximation quality. Sparse representations with discrete weights have been proposed in image and audio compression [38, 131], however with scalar coefficients that are quantized independently and not in the prospect of approximate search. We propose a novel approach that serves our aim better, namely employing vector quantization of coefficient vectors $\hat{\boldsymbol{\alpha}}$. These vectors are of modest size, *i.e.*, M is between 4 and 16 in our experiments. Classical k-means clustering is thus well adapted to produce a codebook $A = [\mathbf{a}_1 \cdots \mathbf{a}_P]_{M \times P}$ for their quantization. This is done after the main dictionaries have been learned.[2]

Denoting $p(\boldsymbol{\alpha}) = \operatorname{argmin}_{p \in [\![1,P]\!]} \|\boldsymbol{\alpha} - \mathbf{a}_p\|$ the index of the vector-quantization of $\boldsymbol{\alpha}$ with this codebook, the final approximation of vector $\mathbf{x}$ reads:

$$Q(\mathbf{x}) = C(\mathbf{k})\mathbf{a}_{p(\hat{\alpha})}, \tag{3.7}$$

with $\mathbf{k}$ function of $\mathbf{x}$ (Eq. 3.3) and $\hat{\boldsymbol{\alpha}} = C(\mathbf{k})^\dagger \mathbf{x}$ (Eq. 3.4) function of $\mathbf{k}$ and $\mathbf{x}$.

Code size A key feature of structured quantization is that it provides the approximation accuracy of extremely large codebooks while limiting learning, coding and search complexities: The M codebooks of size K are as expensive to learn and use as a single codebook of size MK but give effectively access to K^M codewords. In the typical setting where $M = 8$ and $K = 256$, the effective number of possible encodings is 2^{64}, that is more than 10^{19}. This 64-bit encoding capability is obtained by learning and using only 8-bit quantizers. Similarly, quantized sparse coding offers up to $K^M \times P$ encoding vectors, which amounts to $M \log_2 K + \log_2 P$ bits. Structured quantization with $M + 1$ codebooks, all of size K except one of size P has the same code-size, but leads to a different discretization of the ambient vector space $\mathbb{R}^D$. The aim of the experiments will be to understand how trading part of the vector encoding budget for encoding jointly the scalar weights can benefit approximate search.

3.3 Sparse coding extension of PQ and RVQ

In the absence of specific constraints on the M dictionaries, the proposed quantized sparse coding can be seen as a generalization of residual vector quantization (RVQ), with linear combinations rather than only sums of centroids. Hierarchical code structure and search methods (see Section 3.4 below) are analog. To highlight this relationship, we will denote "Qα-RVQ" the proposed encoder.

In case dictionaries are constrained to stem from predefined orthogonal sub-spaces V_ms such that $\mathbb{R}^D = \bigoplus_{m=1}^M V_m$, the proposed approach simplifies notably. Encoding vectors

[2] Alternate refinement of the vector dictionaries C^ms and of the coefficient codebook A led to no improvement. A possible reason is that dictionaries update does not take into account that the coefficients are vector quantized, and we do not see a principled way to do so.

and learning dictionaries can be done independently in each subspace, instead of in sequence. In particular, when each subspace is spanned by D/M (assuming M divides D) successive canonical vectors, *e.g.*, $V_1 = \mathsf{span}(\mathbf{e}_1 \cdots \mathbf{e}_{D/M})$, our proposed approach is similar to product quantization (PQ), which it extends through the use of quantized coefficients. We will denote "Qα-PQ" our approach in this specific set-up: all vectors are partitioned into M sub-vectors of dimension D/M and each sub-vector is approximated independently, with one codeword in PQ, with the multiple of one atom in Qα-PQ.

Algorithm 1 Learning Qα-RVQ

Input: $\mathbf{z}_{1:R}$
Ouput: $C^{1:M}, A$
$\mathbf{r}_{1:R} \leftarrow \mathbf{z}_{1:R}$
for $m = 1 \cdots M$ **do**
    $C^m \leftarrow$ SPHER_K-MEANS($\mathbf{r}_{1:R}$)
    for $r = 1 \cdots R$ **do**
        $k_{m,r} \leftarrow \mathsf{argmax}_{k \in [\![1,K]\!]} \mathbf{r}_r^\top \mathbf{c}_k^m$
        $\mathbf{r}_r \leftarrow \mathbf{r}_r - (\mathbf{r}_r^\top \mathbf{c}_{k_{m,r}}^m)\mathbf{c}_{k_{m,r}}^m$
for $r = 1 \cdots R$ **do**
    $\boldsymbol{\alpha}_r \leftarrow [\mathbf{c}_{k_{1,r}}^1 \cdots \mathbf{c}_{k_{M,r}}^M]^\dagger \mathbf{z}_r$
$A \leftarrow$ K-MEANS($\boldsymbol{\alpha}_{1:R}$)

Algorithm 2 Vector encoding with Qα-RVQ

Input: $\mathbf{x}$, $[\mathbf{c}_{1:K}^{1:M}]$, $[\mathbf{a}_{1:P}]$
Output: $\mathbf{k} = [k_{1:M}]$, p
$\mathbf{r} \leftarrow \mathbf{x}$
for $m = 1 \cdots M$ **do**
    $k_m \leftarrow \mathsf{argmax}_{k \in [\![1,K]\!]} \mathbf{r}^\top \mathbf{c}_k^m$
    $\mathbf{r} \leftarrow \mathbf{r} - (\mathbf{r}^\top \mathbf{c}_{k_m}^m)\mathbf{c}_{k_m}^m$
$\boldsymbol{\alpha} \leftarrow [\mathbf{c}_{k_1}^1 \cdots \mathbf{c}_{k_M}^M]^\dagger \mathbf{x}$
$p \leftarrow \mathsf{argmin}_{p \in [\![1,P]\!]} \|\boldsymbol{\alpha} - \mathbf{a}_p\|$

Learning the dictionaries C^ms and the codebook A for Qα-RVQ is summarized in Alg. 1, and the encoding of a vector with them is in Alg. 2. Learning and encoding in the product case (Qα-PQ) are respectively summarized in Algs. 3 and 4, where all training and test vectors are partitioned in M sub-vectors of dimension D/M, denoted with tilde.

3.4 Approximate search

Three related types of nearest neighbor (NN) search are used in practice, depending on how the (dis)similarity between vectors is measured in $\mathbb{R}^D$: minimum Euclidean distance, maximum cosine-similarity or maximum inner-product. The three are equivalent

Algorithm 3 Learning Qα-PQ

Input: $\mathbf{z}_{1:R}$
Output: $\tilde{C}^{1:M}, A$
for $r = 1 \cdots R$ **do**
    $[\tilde{\mathbf{z}}_{1,r}^\top \cdots \tilde{\mathbf{z}}_{M,r}^\top] \leftarrow \mathbf{z}_r^\top$
for $m = 1 \cdots M$ **do**
    $\tilde{C}^m \leftarrow$ SPHER_K-MEANS$(\tilde{\mathbf{z}}_{m,1:R})$
    for $r = 1 \cdots R$ **do**
        $k \leftarrow \mathrm{argmax}_{k \in [\![1,K]\!]} \tilde{\mathbf{z}}_{m,r}^\top \tilde{\mathbf{c}}_k^m$
        $\alpha_{m,r} \leftarrow \tilde{\mathbf{z}}_{m,r}^\top \tilde{\mathbf{c}}_k^m$
$A \leftarrow$ K-MEANS$(\boldsymbol{\alpha}_{1:R})$

Algorithm 4 Vector encoding with Qα-PQ

Input: $\mathbf{x}$, $[\tilde{\mathbf{c}}_{1:K}^{1:M}]$, $[\mathbf{a}_{1:P}]$
Output: $\mathbf{k} = [k_{1:M}]$, p
$[\tilde{\mathbf{x}}_1^\top \cdots \tilde{\mathbf{x}}_M^\top] \leftarrow \mathbf{x}^\top$
for $m = 1 \cdots M$ **do**
    $k_m \leftarrow \mathrm{argmax}_{k \in [\![1,K]\!]} \tilde{\mathbf{x}}_m^\top \tilde{\mathbf{c}}_k^m$
    $\alpha_m \leftarrow \tilde{\mathbf{x}}_m^\top \tilde{\mathbf{c}}_{k_m}^m$
$p \leftarrow \mathrm{argmin}_{p \in [\![1,P]\!]} \|\boldsymbol{\alpha} - \mathbf{a}_p\|$

when all vectors are ℓ_2-normalized. In visual search, classical descriptors (either at local level or image level) can be normalized in a variety of ways, *e.g.*, ℓ_2, ℓ_1 or blockwise ℓ_2, exactly or approximately.

With cosine-similarity (CS) for instance, the vector closest the query $\mathbf{y}$ in the database $\mathcal{X}$ is $\mathrm{argmax}_{\mathbf{x} \in \mathcal{X}} \frac{\mathbf{y}^\top \mathbf{x}}{\|\mathbf{x}\|}$, where the norm of the query is ignored for it has no influence on the answer. Considering approximations of database vectors with existing or proposed methods, approximate NN (aNN) search can be conducted without approximating the query (asymmetric aNN [66]):

$$\mathrm{CS-aNN}: \quad \operatorname*{argmax}_{\mathbf{x} \in \mathcal{X}} \frac{\mathbf{y}^\top Q(\mathbf{x})}{\|Q(\mathbf{x})\|}. \tag{3.8}$$

As with structured encoding schemes, the form of the approximation in (3.7) permits to expedite the search. Indeed, for $\mathbf{x}$ encoded by $(\mathbf{k}, p) \in [\![1, K]\!]^M \times [\![1, P]\!]$, the approximate cosine-similarity reads

$$\frac{\mathbf{y}^\top C(\mathbf{k}) \mathbf{a}_p}{\|C(\mathbf{k})\mathbf{a}_p\|}, \tag{3.9}$$

where the M inner products in $\mathbf{y}^\top C(\mathbf{k})$ are among the MK ones in $\mathbf{y}^\top C$, which can be computed once and stored for a given query. For each database vector $\mathbf{x}$, computing the numerator then requires M look-ups, M multiplications and $M-1$ sums. We discuss the denominator below.

In the Qα-PQ setup, as the M unit atoms involved in $C(\mathbf{k})$ are mutually orthogonal, the denominator is equal to $\|\mathbf{a}_p\|$, that is one among P values that are independent of the queries and simply pre-computed once for all. In Qα-RVQ however, as in other quantizers with non-orthogonal codebooks, the computation of

$$\|C(\mathbf{k})\mathbf{a}_p\| = \Big(\sum_{m,n=1}^{M} a_{mp} a_{np} \mathbf{c}_{k_m}^{m\top} \mathbf{c}_{k_n}^{n} \Big)^{1/2} \tag{3.10}$$

constitutes an overhead, with $\mathbf{a}_p = [a_{mp}]_{m=1}^{M}$. Two methods are suggested in [5] to handle this problem. The first one consists in precomputing and storing for look-up all inter-dictionary inner products of atoms, *i.e.*, $C^\top C$. For a given query, the denominator can then be computed with $\mathcal{O}(M^2)$ operations. The second method is to compute the norms for all approximated database vectors and to encode them with a non-uniform scalar quantizer (typically with 256 values) learned on the training set. This adds an extra byte to the database vector encoding but avoids the search time overhead incurred by the first method. This computational saving is worth the memory expense for very large scale systems (See experiments on 1 billion vectors in the next section).

Using the Euclidean distance instead of the cosine similarity, *i.e.*, solving $\mathrm{argmin}_{\mathbf{x}\in\mathcal{X}} \left\{ \|Q(\mathbf{x})\|^2 - 2\, \mathbf{y}^\top Q(\mathbf{x}) \right\}$ leads to very similar derivations. The performance of the proposed framework is equivalent for these two popular metrics.

3.5 Experiments

We compare on various datasets the proposed methods, Qα-RVQ and Qα-PQ, to the structured quantization techniques they extend, RVQ and PQ respectively. We use three main datasets: SIFT1M [65], GIST1M [66] and VLAD500K [3].[3] For PQ and Qα-PQ on GIST and VLAD vectors, PCA rotation and random coordinate permutation are applied, as they have been shown to improve performance in previous works. Each dataset includes a main set to be searched ($\mathcal{X}$ of size U), a training set ($\mathcal{Z}$ of size R) and S query vectors. These sizes and input dimension D are given in Table 3.1.

As classically done, we report performance in terms of recall@R, *i.e.*, the proportion of query vectors for which the true nearest neighbor is present among the R nearest neighbors returned by the approximate search.

Introducing coefficients Before moving to the main experiments, we first investigate how the key idea of including scalar coefficients into structured quantization allows more accurate vector encoding. To this end, we compare average reconstruction errors, $\frac{1}{U}\sum_{\mathbf{x}\in\mathcal{X}} \|\mathbf{x} - Q(\mathbf{x})\|_2^2$, obtained on the different datasets by RVQ (resp. PQ) and

[3] VLAD vectors, as kindly provided by Relja Arandjelović, are PCA-compressed to 128 dimensions and unit ℓ_2-normalized; SIFT vectors are 128-dimensional and have almost constant ℓ_2-norm of 512, yielding almost identical nearest-neighbors for cosine similarity and ℓ_2 distance.

Dataset	D	R	U	S
SIFT1M	128	100K	1M	10K
GIST1M	960	500K	1M	1K
VLAD500K	128	400K	0.5M	1K

Table 3.1: Datasets. Data dimension and training/database/query set size of the datasets we use.

the proposed approach *before vector quantization of coefficient vector*, which we denote α-RVQ (resp. α-PQ), see Fig. 3.1. Three structure granularities are considered, $M = 4,\ 8$ and 16. Note that in RVQ and α-RVQ, increasing the number of layers from M to $M' > M$ simply amounts to resuming recursive encoding of residuals. For PQ and α-PQ however, it means considering two different partitions of the input vectors: the underlying codebooks/dictionaries and the resulting encodings have nothing in common.

Reconstruction errors (distortions) are also reported for $K = 2^8$ and 2^{12} respectively. For a given method, reconstruction error decreases when M or K increases. Also, as expected, α-RVQ (resp. α-PQ) is more accurate than RVQ (resp. PQ) for the same (M, K). As we shall see next, most of this accuracy gain is retained after quantizing, even quite coarsely, the coefficient vectors.

Quantizing coefficients Figure 3.2 shows the effect of this quantization on the performance, in comparison to no quantization (sparse encoding) and to classic structured quantization without coefficients. For these plots, we have used one byte encoding for $\boldsymbol{\alpha}$, *i.e.*, $P = 256$, along with $M \in \{4, 8, 16\}$ and $K = 256$. With this setting, Qα-RVQ (resp. Qα-PQ) is compared to both α-RVQ and RVQ (resp. α-PQ and PQ) with the same values of M and K. This means in particular that Qα-RVQ (resp. Qα-PQ) benefits from one extra byte compared to RVQ (resp. PQ). Note that allowing one more byte for RVQ/PQ encoding would significantly increase its learning, encoding and search practical complexities.

Since $\boldsymbol{\alpha}$ has M dimensions, its quantization with a single byte gets cruder as M increases, leading to a larger relative loss of performance as compared to no quantization. For $M = 4$, one byte quantization suffices in both structures to almost match the good performance of unquantized sparse representation. For $M = 16$, the increased degradation remains small within Qα-RVQ. However it is important with Qα-PQ: for a small budget allocated to the quantization of $\boldsymbol{\alpha}$, it is even outperformed by the PQ baseline. This observation is counter-intuitive (with additional information, there is a loss). The reason is that the assignment is greedy: while the weights are better approximated w.r.t. a square loss, the vector reconstruction is inferior with Eq. (3.2). A non-greedy exploration strategy as in AQ would address this problem but would also dramatically increase the assignment cost. This suggests that the size P of the codebook associated with $\boldsymbol{\alpha}$ should

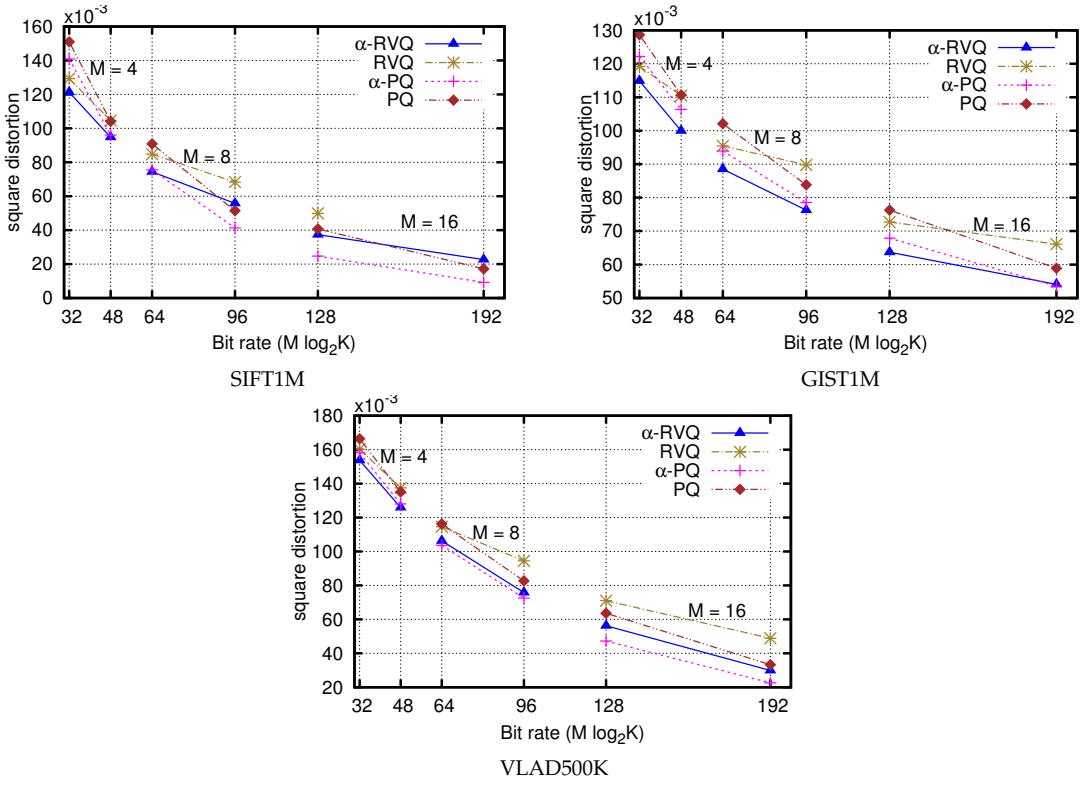


Figure 3.1: Accuracy of structured encoding, with and without coefficients. Squared reconstruction errors produced by structured encoding (PQ and RVQ) and proposed sparse encoding extensions (α-PQ and α-RVQ). For each method, $M = 4, 8, 16$ and $\log_2 K = 8, 12$ are reported. In absence of coefficient quantization here, each code has $M \log_2 K$ bits, *i.e.*, 64 bits for $(M, K) = (8, 256)$.

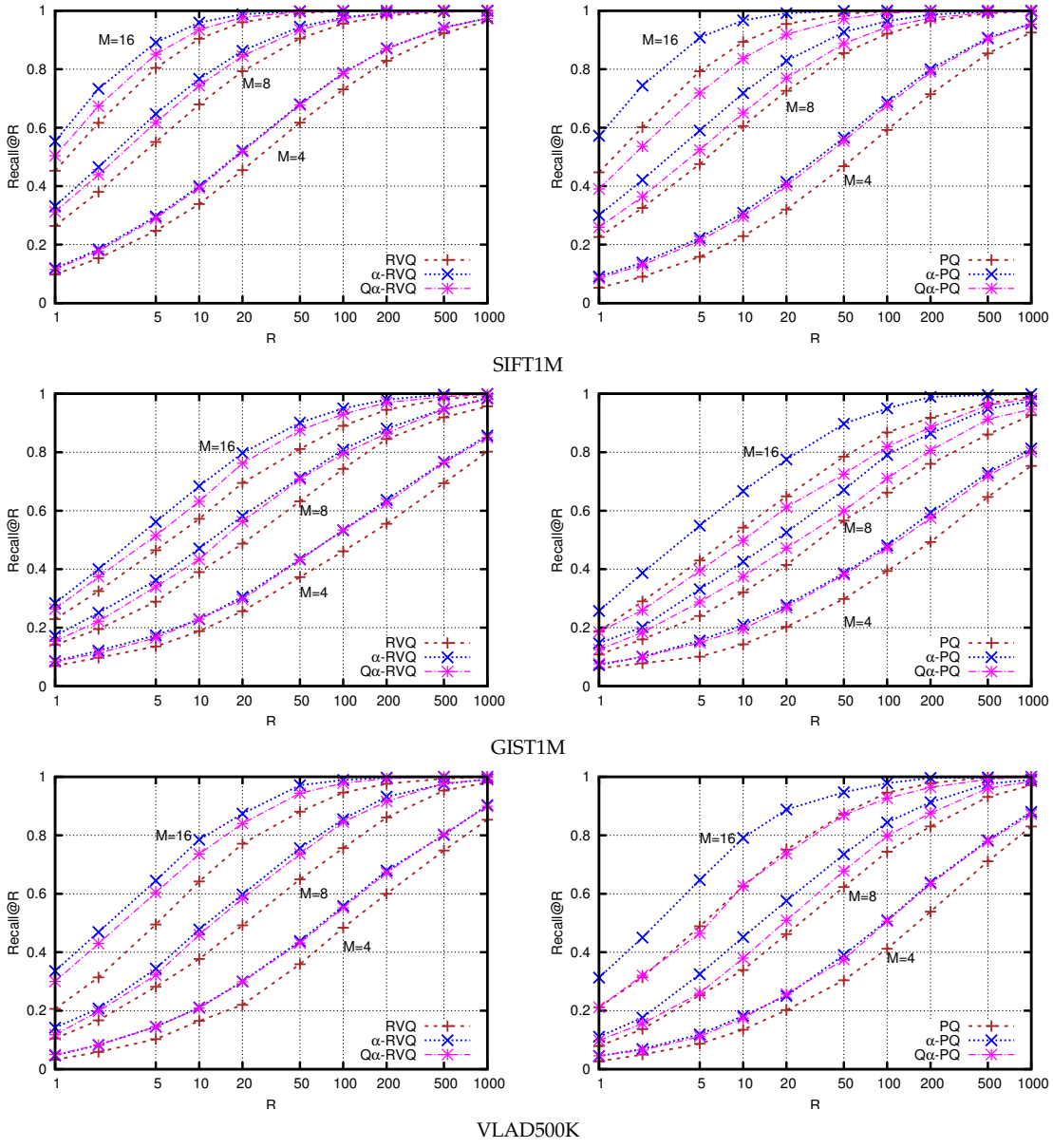


Figure 3.2: Impact of 1-byte α quantization on performance. Recall@R curves for Qα-RVQ, α-RVQ and RVQ (resp. Qα-PQ, α-PQ and PQ) on the three datasets, with $M \in \{4, 8, 16\}$, $K = 256$ and $P = 256$.

	Qα-RVQ vs. RVQ	Qα-PQ vs. PQ
	SIFT/GIST/VLAD	SIFT/GIST/VLAD
$M = 4$	1/1/1	3/3/2
$M = 8$	1/1/1	6/5/5
$M = 16$	1/1/1	15/11/12

Table 3.2: Minimum bits for coefficient encoding. Number of bits required to encode the coefficients *i.e.*, $\log_2 P$ to lower the reconstruction error compared to the corresponding structured quantization on the three datasets, with $M \in \{4, 8, 16\}$ and $K = 256$.

be adapted to the number M of layers.

Hereafter, in Table 3.2, we measure for each dataset the minimum number of bits that must be dedicated to coefficients quantization ($\log_2 P$) to ensure that the reconstruction error with structured sparse coding remains below the one of the corresponding structured quantization method. Interestingly, the first bit allocated to α improves upon RVQ for all the settings and datasets. In contrast and as discussed before, for Qα-PQ, more bits must be allocated to the weights for larger M to guarantee a better representation. For instance, an overhead of 6 bits is required for $M = 8$.

Comparing at fixed code size For large scale search, considering (almost) equal encoding sizes is a good footing for comparisons. This can be achieved in different ways. In the case of RVQ and Qα-RVQ, the recursive nature of encoding provides a natural way to allocate the same encoding budget for the two approaches: we compare Qα-RVQ with (M, K, P) to RVQ with M codebooks of size K and a last one of size P. For PQ and Qα-PQ, things are less simple: adding one codebook to PQ to match the code size of Qα-PQ leads to a completely different partition of vectors, creating new possible sources of behavior discrepancies between the two compared methods. Instead, we compare PQ with M codebooks of size K to Qα-PQ with M dictionaries of size $K/2$ and $P = 2^M$ codewords for coefficient vectors. This way, vector partitions are the same for both, as well as the corresponding code sizes ($M \log_2 K$ bits for PQ and $M \log_2 \frac{K}{2} + \log_2 2^M = M \log_2 K$ bits for Qα-PQ).

Sticking to these rules, we shall compare next structured quantization and quantized sparse representation for equal encoding sizes.

CS-aNN We compare RVQ to Qα-RVQ and PQ to Qα-PQ for different code sizes, from 8 to 24 bytes per vector, on the task of maximum cosine similarity over ℓ_2-normalized vectors. Corresponding Recall@1 curves are in Fig. 3.3. Qα-RVQ clearly outperforms RVQ on all datasets, even with a substantial margin on GIST1M and VLAD500K, *i.e.*, around 30% relative gain at 24 bytes. The comparison between PQ and Qα-PQ leads to mixed conclusions: while Qα-PQ is below PQ on SIFT1M, it is slightly above for GIST1M and almost similar for VLAD500K. Note however that, for the same number $M \log_2 K$ of

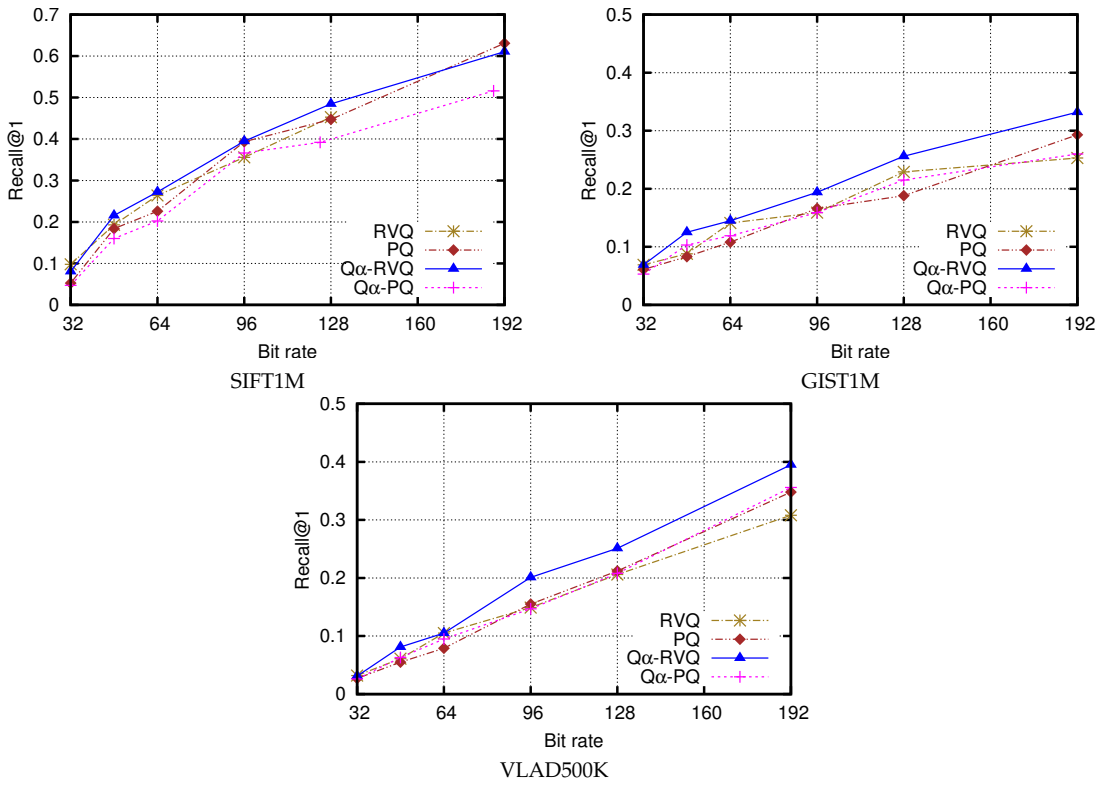


Figure 3.3: Comparative CS-aNN performance for different encoding sizes. Recall@1 on the three datasets for increasing number of encoding bits, comparing PQ and RVQ with Qα-PQ and Qα-RVQ respectively.

	SIFT		GIST	
encoding bits	64	72	64	72
PQ	23515	20054	0.7121	0.6733
Qα-PQ	25859	22007	0.7224	0.6868
RVQ	22170	20606	0.6986	0.6734
Qα-RVQ	**22053**	**19976**	**0.6537**	**0.6174**

Table 3.3: Comparative distortions in Euclidean aNN setting. Average squared reconstruction errors on un-normalized SIFT1M and GIST1M.

encoding bits, Qα-PQ uses $M\frac{K}{2} + 2^M$ centroids, which is nearly half the number MK of centroids used by PQ in low M regimes (*e.g.*, when $K = 256$, 528 vs. 1024 centroids for $M = 4$ and 1280 vs. 2048 centroids for $M = 8$). Much fewer centroids for equal code size and similar performance yield computational savings in learning and encoding phases.

Euclidean aNN In order to conduct comparison with other state-of-art methods such as extensions of PQ and of RVQ, we also considered the Euclidean aNN search problem, with no prior normalization of vectors. For this problem, the proposed approach applies similarly since the minimization problem $\operatorname{argmin}_{\mathbf{x}\in\mathcal{X}} \|\mathbf{y} - Q(\mathbf{x})\|^2 = \operatorname{argmax}_{\mathbf{x}\in\mathcal{X}} \mathbf{y}^\top Q(\mathbf{x}) - \frac{\|Q(\mathbf{x})\|^2}{2}$ involves the same quantities as the one in (3.8).

Recall@R curves are provided in Fig. 3.4 on two of the three datasets, relying on results reported in [99] for CKM, RVQ and ERVQ, and [5], [135] for AQ and CQ respectively. We observe again that Qα-PQ is below PQ on SIFT but on par with it on GIST. On SIFT, Qα-RVQ, ERVQ and CQ perform similarly, while on GIST Qα-RVQ outperforms all, including CQ and ERVQ. As discussed in Section 3.1, AQ is the most accurate but has very high encoding complexity. CQ also has higher encoding complexity compared to our simple and greedy approach.

Table 3.3 shows reconstruction errors for the same setting as in Fig. 3.4. This is consistent with the results in Fig. 3.4, and shows again that Qα-RVQ is the best performer and that Qα-PQ does not improve on PQ with the same number of encoding bits.

Note that the lower performance of Qα-PQ compared to PQ is because it uses half the number of codewords to have equal or smaller memory footprint. Relative timings in Tab. 3.4 indicate Qα-PQ is substantially faster for learning and encoding in this setting. Our methods are slower in search but this overhead has minimal effect in the applications with very large scale data as we shall see in our billion-scale experiments.

Table 3.5 provides recall rates for various PQ based methods on SIFT1M with 64 bits encoding. CKM and OPQ are very similar extensions on PQ and thus perform similarly. The improvement provided by CKM/OPQ on PQ is complimentary to Qα-PQ. By using OPQ instead of PQ within Qα-PQ, calling it Qα-OPQ, we get similar gains as OPQ gives over PQ. This can be seen by comparing the gains of Qα-OPQ over Qα-PQ and OPQ

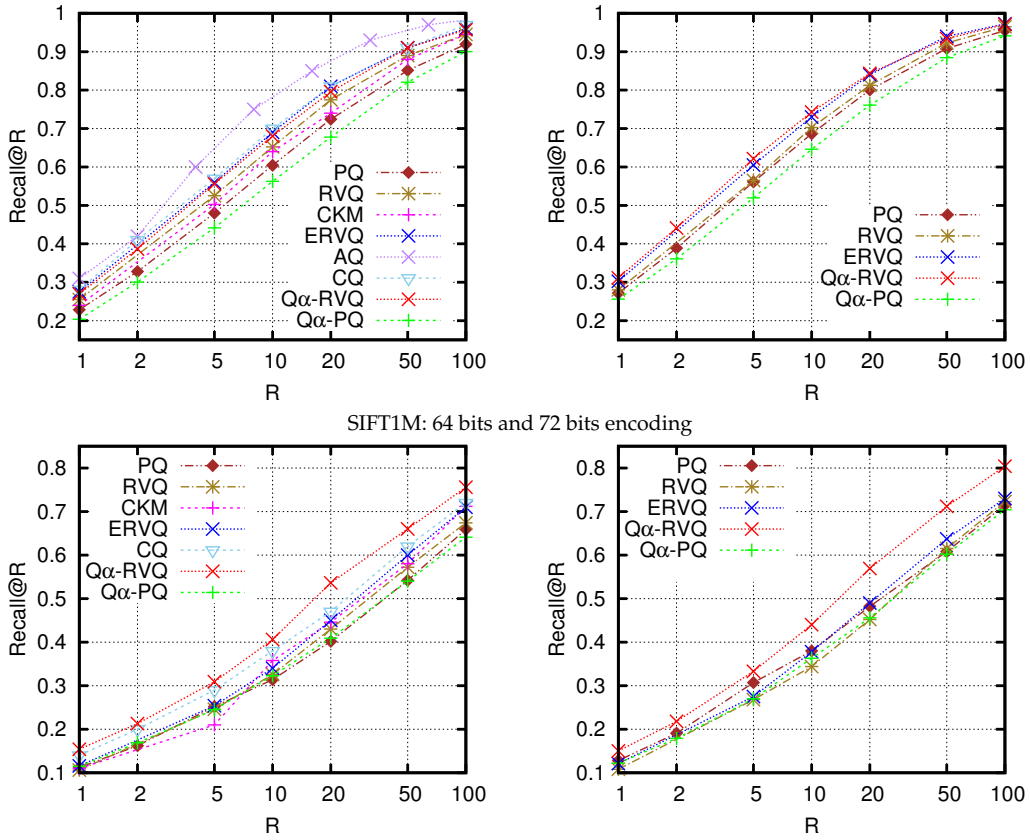


Figure 3.4: Performance comparison for Euclidean-aNN. Recall@R curves on SIFT1M and GIST1M, comparing proposed methods to PQ, RVQ and to some of their extensions, CKM [99], ERVQ [1], AQ [5] and CQ [135].

	PQ	Qα-PQ	RVQ	Qα-RVQ
learn	1	0.212	1.250	0.719
encode	1	0.206	1.347	0.613
search	1	1.867	1.220	1.909

Table 3.4: Relative timings. Learning, encoding and search timings w.r.t. PQ on SIFT1M with 64 bits encoding. Qα-PQ and Qα-RVQ have faster learning/encoding as they use inner product instead of ℓ_2 distance and have fewer codewords.

Recall	PQ	CKM	OPQ	Qα-PQ	Qα-OPQ
R@1	0.228	0.240	0.243	0.204	0.227
R@10	0.604	0.640	0.638	0.562	0.603
R@100	0.919	0.945	0.942	0.900	0.927

Table 3.5: CKM/OPQ comparison with PQ. Performance of PQ based methods on SIFT1M with 64 bits encoding. $(M, K) = (8, 256)$ for PQ, CKM and OPQ and $(M, K, P) = (8, 128, 256)$ for our methods.

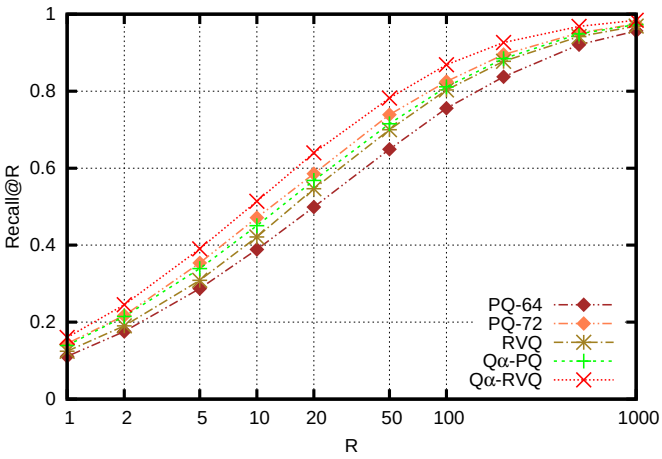
over PQ. These results of OPQ and Qα-OPQ are not plotted in 3.4 to maintain clarity.

Very large scale experiments on BIGANN We validate our approach on large scale experiments carried out on the BIGANN dataset [68], which contains 1 billion SIFT vectors ($U = 1B$, $R = 1M$ out of the original 100M training set and $S = 10K$ queries). At that scale, an inverted file (IVF) system based on a preliminary coarse quantization of vectors is required (as explained in Section 2.5). In our experiments, each vector is quantized over $N = 8192$ centroids, and it is its residual relative to assigned centroid that is fed to the chosen encoder. At search time, the query is multiply assigned to its $W_c = 64$ closest centroids and W_c searches are conducted over the corresponding vector lists (each of average size U/N). Performance is reported in Fig. 3.5 for PQ, RVQ and their proposed extensions. For all of them the setting is $M = 8$ and $K = 256$, except for PQ-72 bits ($K = 512$). All of them use the exact same IVF structure, which occupies approximately 4GB in memory (4B per vector). For RVQ and Qα-RVQ, norms of approximated database vectors are quantized over 256 scalar values.

The best performance is obtained with the proposed Qα-RVQ approach, which requires 10 bytes per vector, thus a total of 14GB for the whole index. The second best aNN search method is PQ-72 bits, which requires 9 bytes per vector, hence 13GB of index. While both indexes have similar sizes and fit easily in main memory, PQ-72 relies on twice as many vector centroids which makes learning and encoding more expensive.

The superior performance of Qα-RVQ comes at the price of a 70% increase of search time per query compared to PQ. This can nonetheless be completely compensated for since the hierarchical structure of Qα-RVQ lends itself to additional pruning after the one with IVF. The W'_c atoms most correlated with the query residual in C^1 are determined, and dataset vectors whose first layer encoding uses none of them are ignored. For $W'_c = 128$, search time is reduced substantially, making Qα-RVQ 10% faster than PQ, with no performance loss (hence superior to PQ-72). A more drastic pruning ($W'_c = 8$) makes performance drop below that of PQ-72, leaving it on par with PQ-64 while being almost 6 times faster.

A variant of IVF, called "inverted multi-index" (IMI) [4] is reported to outperform IVF in speed and accuracy, by using two-fold product quantization instead of vector quantization to produce the first coarse encoding. Using two codebooks of size N, one for each



Method (b)	R@1	R@10	R@100	time
PQ-64 (8)	0.111	0.388	0.756	1.00
PQ-72 (9)	0.144	0.471	0.825	1.03
RVQ (9)	0.124	0.421	0.803	1.02
Qα-PQ (9)	0.139	0.450	0.811	1.69
Qα-RVQ (10)	0.160	0.514	0.868	1.72
Qα-RVQ$_{128}$	0.160	0.514	0.868	0.89
Qα-RVQ$_{8}$	0.151	0.467	0.730	0.17

Figure 3.5: Large scale performance with IVF. Recall@R on the BIGANN 1B-SIFT dataset and 10K queries. For all methods, $M = 8$ and $K = 256$, except for "PQ-72" ($K = 512$). For quantized sparse coding methods, $P = 256$ and norms in residual variant are quantized over 256 scalar values, resulting encoding sizes (b) being given in bytes per vector. All methods share the same IVF index with $N = 2^{13}$ and $W_c = 64$. Subscripted Qα-RVQ denotes variants with additional pruning ($W'_c = 128$ and 8 resp.). Search timings are expressed relative to PQ-64.

half of the vectors, IMI produces N^2 inverted lists. We have run experiments with this alternative inverted file system, using $N = 2^{14}$ and scanning a list of $T = 100K$, $30K$ or $10K$ vectors, as proposed in [4]. The comparisons with PQ-64 based on the same IMI are summarized in Tab. 3.6 in terms of recall rates and timings. For all values of T, the proposed Qα-RVQ and Qα-PQ perform the best and with similar search time as RVQ and PQ-64. Also, Qα-RVQ with $T = 30K$ has the same recall@100 as PQ-64 with $T = 100K$ while being twice as fast (14ms vs. 29ms per query). For a fixed T, PQ-64 and Qα-PQ (resp. RVQ and Qα-RVQ) have the same search speed, as the overhead of finding the T candidates and computing look-up tables dominates for such relatively short lists. The T candidates for distance computation are very finely and scarcely chosen. Therefore, increasing the size K of dictionaries/codebooks in the encoding method directly affects search time. This advocates for our methods, as for equal (M, K) and an extra byte for encoding coefficients, Qα-RVQ and Qα-PQ always give better performance. Compared to PQ-72, Qα-PQ is faster (only half the number of codewords is required in the quantization) with slightly lower accuracy. Qα-RVQ is more accurate with extra execution time compared to PQ-72.

	T =100K				T =30K				T =10K			
Method (b)	R@1	R@10	R@100	time	R@1	R@10	R@100	time	R@1	R@10	R@100	time
PQ-64 (8)	0.170	0.535	0.869	29	0.170	0.526	0.823	11	0.166	0.495	0.725	5
RVQ (9)	0.181	0.553	0.877	37	0.180	0.542	0.831	14	0.174	0.506	0.729	8
Qα-PQ (9)	0.200	0.587	0.898	30	0.198	0.572	0.848	11	0.193	0.533	0.740	5
Qα-RVQ (10)	0.227	0.630	0.920	37	0.225	0.613	0.862	14	0.217	0.566	0.747	8
PQ-72 (9)	0.207	0.603	0.902	34	0.205	0.586	0.849	12	0.2	0.547	0.739	6

Table 3.6: Performance and timings with IMI on 1B SIFTs. Recalls are reported along with search time in milliseconds per query as a function of the length T of candidate list to be exhaustively scanned. For each method, the encoding size (b) is given in bytes per vector.

3.6 Discussion and conclusion

In this chapter we present a novel quantized sparse representation that is specially designed for large scale approximate nearest neighbour search. The residual form of this representation, Qα-RVQ, clearly outperforms RVQ in all datasets and settings, for equal code size. Within the recursive structure of residual quantization, the introduction of additional coefficients in the representation thus offers accuracy improvements that translate into aNN performance gains, even after drastic vector quantization of these coefficients. One possible reason for the proposed approach to be especially successful in its residual form lies in the rapid decay of the coefficients that the hierarchical structure induces. This facilitates quantization of coefficient vectors, even with 1 byte only. In its partitioned variant, this property is not true anymore, and the other proposed approach, Qα-PQ, brings less gain. It does however improve over PQ for image-level descriptors (GIST and VLAD), especially in small M regimes, while using fewer centroids.

As demonstrated on the billion-size BIGANN dataset, the proposed framework can be combined with existing inverted file systems like IVF or IMI to provide highly competitive performance on large scale search problems. In this context, we show in particular that both Qα-PQ and Qα-RVQ offer higher levels of search quality compared to PQ and RVQ for similar speed and that they allow faster search with similar quality. Regarding Qα-RVQ, it is also worth noting that its hierarchical structure allows one to prune out most distant vectors based only on truncated descriptors, as demonstrated on BIGANN within IVF system. Conversely, this nested structure permits to refine encoding if desired, with no need to retrain and recompute the encoding up to the current layer.

On a different note, the successful deployment of the proposed quantized sparse encoding over million to billion-sized vector collections suggests it could help scaling up sparse coding massively in other applications.

CHAPTER

FOUR

SUBIC: A SUPERVISED, STRUCTURED BINARY CODE FOR IMAGE SEARCH

In this chapter, we propose a *supervised structured binary code* to encode images for efficient large scale image search. The structured binary code is produced by a supervised deep convolutional neural network by employing a novel block-softmax nonlinearity and batch-based entropy losses. This chapter is based on the following publication:

SUBIC: A supervised, structured binary code for image search. H. Jain, J. Zepeda, P. Pérez, and R. Gribonval. In Proceedings of the IEEE *International Conference on Computer Vision* (ICCV), 2017.

Deep convolutional neural networks (CNNs) have proven to be versatile image representation tools with great generalization power, a quality that has rendered them indispensable in image search. A given network trained on the ImageNet dataset [30], for example, can achieve excellent performance when transferred to a variety of other datasets [45, 77], or even to other visual search tasks [106]. This quality of *transferability* is important in large-scale image search, where the time or resources to compile annotations in order to train a new network for every new dataset or task are not available.

A second desirable property of image representations for large-scale visual search is that of being *compact yet functional*. A paramount example of such a representation is provided by image indexing schemes, such as Product Quantization (PQ) [66] and others, that rely on vector quantization with structured, unsupervised codebooks [5, 24, 135]. PQ can be seen as mapping a feature vector into a binary vector consisting of a concatenation of one-hot (a binary vector with all entries but one being zero) encoded codeword indices. One can directly compare an uncompressed query feature with these binary vectors by means of an inner product between the binary vectors and a real-valued mapping of the query feature vector.

It is not surprising that, with the dawn of the deep learning revolution, many recent research efforts have been directed towards supervised learning of deep networks that produce compact and functional binary features [27, 31, 78, 80, 82, 129, 134, 137]. One

commonality between these approaches – which we refer to collectively as *deep hashing* methods – is their reliance on element-wise binarization mechanisms consisting of either sigmoid/tanh non-linearities [27, 78, 80, 82, 129, 134, 137] or element-wise binarizing penalties such as [27, 31]. Indeed, to our knowledge, ours is the first approach to impose a structure on the learned binary representation: We employ two entropy-based losses to induce a one-hot block structure in the produced binary feature vectors, while favoring statistical uniformity in the support of the active bits of each block. The resulting structured binary code has the same structure as a PQ-encoded feature vector.

Imposing structure on the support of the binary representation has two main motivations: First, structuring allows a better exploitation of the binary representation's support to encode semantic information, as exemplified by approaches that learn to encode face parts [10], visual attributes [90] and text topics [69] in the support of the representation under weak supervision. We promote this desirable property by means of an entropy-based loss that encourages uniformity in the position of the active bits – a property that would not be achievable using a simple softmax non-linearity. Note that, as a related added benefit, the structuring makes it possible to use a binary representation of larger size without incurring extra storage. Second, the structuring helps in regularizing the architecture, further contributing to increased performance relative to other, non-structured approaches.

While all previously existing deep hashing methods indeed produce very compact, functional representations, they have not been tested for transferability. The main task addressed in all these works is that of category retrieval wherein a given test example is used to rank all the test images in all classes. Yet all deep hashing approaches employ a *single-domain* approach wherein the test classes and training classes are the same. It has been established experimentally [108] that excellent performance can be achieved in this test by simply assigning to each stored database image, the class label produced by a classifier trained on the corresponding training set. Hence, it is also important to test for *cross-domain* category retrieval, wherein the architecture learned on a given set of training classes is tested on a new, disjoint set of test classes. We present experiments of both types in this work, outperforming several baselines in the cross-domain test and recent deep hashing methods in the single-domain test.

The contributions of the present work can be summarized as follows:

- We introduce a simple, trainable, CNN layer that encodes images into structured binary codes that we coin SUBIC. While all other approaches to supervised binary encoding use element-wise binarizing operations and losses, ours are block-based.
- We define two block-wise losses based on code entropy that can be combined with a standard classification loss to train CNNs with a SUBIC layer.
- We demonstrate that the proposed binary features outperform the state-of-the art

in single-domain category retrieval, two competitive baselines in cross-domain category retrieval and image classification, and state-of-the art unsupervised quantizers in image retrieval.

- Our approach enables asymmetric search with a search complexity comparable to that of deep hashing.

4.1 Related work

Method	supervision	binarization (train – test)	code-based loss on	base CNN training	cross-domain
CNNH+ [129]	pair-wise	sigmoid–threshold	dist. to target code	yes	no
DRSCH [134]	triplet-wise	sigmoid–threshold	none	yes	no
DSRH [137]	triplet-wise	sigmoid–threshold	training average	yes	no
DNNH [78]	triplet-wise	sigmoid–threshold	none	yes	no
DLBHC [80]	point-wise	sigmoid–threshold	none	fine-tuning	no
DSH [82]	pair-wise	sigmoid–threshold	distance to binary	yes	no
BDNN [31]	pair-wise	built-in	feature reconst. error	no	no
SUBIC (ours)	point-wise	block softmax–argmax	block-based entropies	yes	yes

Table 4.1: Comparison of proposed approach to recent supervised binary hashing techniques.

We discuss here the forms of vector quantization and binary hashing that are the most important for efficient visual search with compact codes, and we explain how our approach relates to them.

Unsupervised structured quantization. Vector quantization (VQ), *e.g.* with unsupervised k-means, is a classic technique to index multi-dimensional data collections in a compact way while allowing efficient (approximate) search. Structured versions of VQ, *e.g.* product, additive or composite [5, 39, 40, 60, 66, 73, 99, 135], have established impressive indexing systems for large scale image collections, as discussed in Chapter 2. Coupled with single or multiple index inverted file systems [4, 68], these VQ techniques currently offer state-of-the-art performance for very large-scale high-dimensional nearest-neighbor search (relative to the Euclidean distance in input feature space) and instance image search based on visual similarity. All these unsupervised quantization techniques operate on engineered or pre-trained image features. Owing to the success of CNNs for image analysis at large, most recent variants use off-the-shelf or specific CNN features as input representation, *e.g.*, [8, 71, 83, 122]. However, contrary to binary hashing methods discussed below, VQ-based indexing has not yet been approached from a supervised angle where available semantic knowledge would help optimize the indexed codes and possibly the input features. In the present work, we take a supervised encoding approach that bears a strong connection to supervised binary hashing, while exploiting an important aspect of these powerful unsupervised VQ techniques, namely the structure of the code. The binary codes produced by our approach are in a discrete product space of size K^M while allowing $\mathcal{O}(M \log K)$ storage and $\mathcal{O}(M)$ search complexity.

Deep, supervised hashing. Binary hashing is a long-standing alternative to the above-mentioned VQ methods, and the deep learning revolution has pushed the state-of-the-art of these approaches. Deep supervised hashing methods – CNNH+ [129], DRSCH [134], DSRH [137], DNNH [78], DLBHC [80], DSH [82] and BDNN [31] – share the following high-level principles. An off-the-shelf or home-brewed convolutional network f_1 is used to extract a high-dimensional distributed representation $\mathbf{x} \in \mathbb{R}^d$ from an input image $\mathbf{I}$. A subsequent fully connected encoding layer f_2 turns this feature vector into a compact binary code $\mathbf{h} \in \{0,1\}^B$ of B bits through final entry-wise thresholding (or sign function for centered codes), B typically ranging from 12 to 64 bits. At training time, this binarization is usually relaxed using a sigmoid (or tanh for centered codes) – with the exception of BDNN [31] –, which results in an encoding layer that outputs vectors in $\in [0,1]^B$. Using semantic supervision, f_2 is trained while f_1 is fixed to pre-trained values, fine-tuned or trained from scratch. Supervision coming from class labels is used either directly (classification training) [80] or using tuples (pairs [31, 82, 129] or triplets [78, 134, 137]) as in metric learning. Table 4.1 summarizes the specifics of each of these methods.

DLBHC [80] is a simple instance employing a sigmoid-activated encoding layer grafted to the pre-trained AlexNet architecture [74] and trained using a standard classification objective. Other methods employ additional training loss(es) at the code level to induce desirable properties. BDNN [31] uses the code-to-feature reconstruction error, making the approach applicable in an unsupervised regime. DSH [82] employs a W-shaped loss with minima at the desired code values, while DSRH [137] penalizes the average of each bit over the training set such that its distribution is approximately centred (final code is in $\{-1,+1\}^B$). CNN+ [129] proposes the direct supervision of hash functions with target binary codes learned in a preliminary phase via low-rank factorization of a full pairwise similarity matrix. Note also that DRSCH [134] learns bit-wise weights along with the binary encoder, which results in richer codes but more costly distances to compute at search time.

As described in detail next, our approach follows the same high-level principles discussed above, but with important differences. The first, key difference lies in the structure of the codes. We define them as the concatenation of M one-hot vectors (binary vectors with all entries but one being zero) of size K. This gives access to K^M distinct codes, hence corresponding to an effective bit-size of $M \log_2 K$ bits, as in VQ methods that combine M codewords, each taken from a codebook of size K. Binarization and its relaxed training version thus operate at the block level. This specific code structure is combined with novel loss terms that enforce respectively the one-sparsity of each block and the effective use of the entire block support. Also, contrary to most supervised binary hashing approaches, with the exception of [80], we resort only to point-wise supervision.

To our knowledge, only one other approach has incorporated product-wise structuring within a deep learning pipeline [18]. Yet that method does not learn the structuring

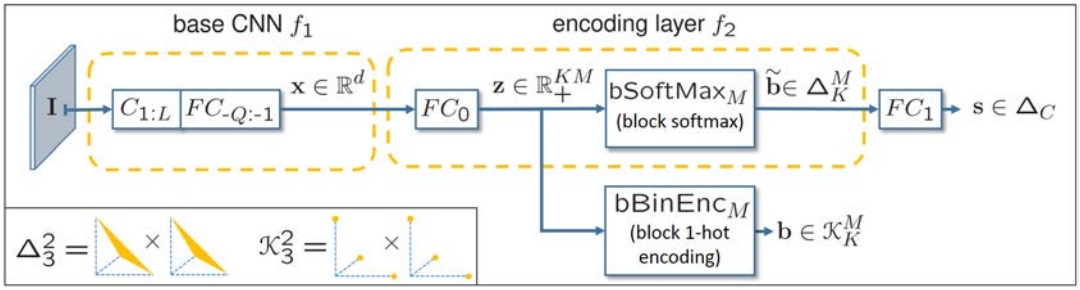


Figure 4.1: Proposed architecture and notations. A feature is extracted from image $\mathbf{I}$ by a base CNN f_1 and binarized using a block-structured encoding layer f_2 consisting of a fully-connected layer followed by a *block softmax* during training, or a *block* 1*-hot encoder* during testing.

as part of a deep architecture, relying rather on a standard product quantizer that is updated once per epoch in an unsupervised manner.

4.2 Approach

We describe in this section the design of our SUBIC architecture, its supervised training and its use for visual search.

4.2.1 Architecture

Following the approach discussed above, we consider the following classification feed-forward network (Fig. 4.1):

$$\mathbf{s} \triangleq FC_1 \circ f_2 \circ f_1(\mathbf{I}), \tag{4.1}$$

where $\mathbf{I}$ is an input image, f_1 a deep CNN with L convolutional layers (inc. pooling and normalization, if any) and Q fully-connected layers, f_2 a binary encoding layer, FC_1 a C-class classification layer, and $\mathbf{s}$ the C-dimensional vector of class-probability estimates.

We aim for the binary encoding layer f_2 to produce structured binary vectors $\mathbf{b}$ consisting of the concatenation of M one-hot encoded vectors $\mathbf{b}_m$, $m = 1, \dots, M$, of dimension K, *i.e.*, $\mathbf{b} = [\mathbf{b}_1; \dots; \mathbf{b}_M]$.[1] Formally, the *blocks* $\mathbf{b}_m$ should satisfy

$$\mathbf{b}_m \in \mathcal{K}_K \triangleq \{\mathbf{d} \in \{0,1\}^K \text{ s.t. } \|\mathbf{d}\|_1 = 1\}. \tag{4.2}$$

Accordingly, our codes $\mathbf{b}$ should come from the discrete product set $\mathcal{K}_K^M$.

In practice, f_2 employs a fully-connected layer FC_0 with ReLU non-linearity producing real-valued vectors $\mathbf{z} \in \mathbb{R}_+^{KM}$ likewise consisting of M K-dimensional blocks $\mathbf{z}_m$. A second non-linearity operates on each $\mathbf{z}_m$ to produce the corresponding binarized block. We use a different binarization strategy at training time (top branch in Fig. 4.1) and at test time (bottom branch), as discussed next.

[1] Using vector stacking notation $[\mathbf{a}; \mathbf{b}] = [\mathbf{a}^\top, \mathbf{b}^\top]^\top$, where $\mathbf{a}$ and $\mathbf{b}$ are column-vectors.

Training architecture. Similarly to supervised binary hashing approaches discussed in Section 4.1, we enable back-propagation during our learning process by relaxing the structured binarization constraint (4.2), producing instead structured real-valued codes $\widetilde{\mathbf{b}} \in \Delta_K^M$, where

$$\Delta_K \triangleq \{\mathbf{d} \in [0,1]^K \text{ s.t. } \|\mathbf{d}\|_1 = 1\} \tag{4.3}$$

is the convex hull of $\mathcal{K}_K$ (see Fig. 4.1 bottom-left, for the examples Δ_3^2 and $\mathcal{K}_3^2$). We achieve this by introducing the *block-softmax non-linearity* $\widetilde{\mathbf{b}} = \mathsf{bSoftMax}_M(\mathbf{z})$ (*cf*. Fig. 4.1) which computes the blocks $\widetilde{\mathbf{b}}_m$ from the corresponding blocks $\mathbf{z}_m$ as follows ($\exp(\cdot)$ denotes element-wise exponentiation):

$$\widetilde{\mathbf{b}}_m = \frac{1}{\|\exp(\mathbf{z}_m)\|_1} \exp(\mathbf{z}_m). \tag{4.4}$$

Test time architecture. At test time, the block softmax is replaced (*cf.* Fig. 4.1, bottom branch) by a *block one-hot encoder* $\mathbf{b} = \mathsf{bBinEnc}_M(\mathbf{z})$, which uses $\mathbf{z}$ to efficiently compute the projection of each block $\widetilde{\mathbf{b}}_m \in \Delta_K$ onto $\mathcal{K}_K$ using

$$\mathbf{b}_m[k] = \begin{cases} 1 & \text{if } k = \operatorname{argmax}_r \mathbf{z}_m[r], \\ 0 & \text{otherwise,} \end{cases} \tag{4.5}$$

where $\mathbf{d}[k]$ denotes the k-th entry of a vector $\mathbf{d}$. Note, particularly, that $\mathsf{bBinEnc}_M(\mathbf{z}) = \mathsf{bBinEnc}_M(\widetilde{\mathbf{b}})$.

4.2.2 Supervised loss and training

In order to bring real-valued code vectors $\widetilde{\mathbf{b}}$ as close as possible to block-wise one-hot vectors, while making the best use of coding budget, we introduce two entropy-based losses that will be part of our learning objective. Our approach assumes a standard learning method wherein training examples $(\mathbf{I}^{(i)}, y^{(i)})$ consisting of an image $\mathbf{I}^{(i)}$ and its class label $y^{(i)} \in \{1, \dots, C\}$ are divided into mini-batches $\{(\mathbf{I}^{(i)}, y^{(i)})\}_{i \in \mathcal{T}}$ of size $|\mathcal{T}| = T$.

Our losses will be based on *entropy*, which is computed for a vector $\mathbf{p} \in \Delta_K$ as follows:[2]

$$\mathsf{E}(\mathbf{p}) \triangleq -\sum_{k=1}^{K} \mathbf{p}[k] \log_2 \mathbf{p}[k]. \tag{4.6}$$

Entropy is smooth and convex and further has the interesting property that it is the theoretical minimum average number of bits per symbol required to encode an infinite sequence of symbols with distribution $\mathbf{p}$ [26]. Accordingly, it is exactly zero, its minimum, if $\mathbf{p}$ specifies a deterministic distribution (*i.e.*, $\mathbf{p} \in \mathcal{K}_K$) and $\log_2 K$, its maximum, if it specifies a uniform distribution (*i.e.*, $\mathbf{p} = \frac{1}{K}\mathbf{1}$).

[2] With the usual convention $0 \log_2(0) = 0$.

Toward block-wise one-hot encoding. Given the merits of structured binary codes discussed previously, we aim to produce feature vectors $\widetilde{\mathbf{b}} = [\widetilde{\mathbf{b}}_1; \dots; \widetilde{\mathbf{b}}_M]$ consisting of blocks $\widetilde{\mathbf{b}}_m$ that approximate one-hot encoded vectors, thus that have a small projection error

$$\min_{\mathbf{d} \in \mathcal{K}_K} \|\mathbf{d} - \widetilde{\mathbf{b}}_m\|_2. \tag{4.7}$$

In the ideal case where $\widetilde{\mathbf{b}}_m \in \mathcal{K}_K$, it has minimum entropy of 0. The convexity/smoothness of $\mathsf{E}(\cdot)$ means that blocks with low entropy will have small projection error (4.7), thus suggesting penalizing our learning objective for a given training image using $\sum_m \mathsf{E}(\widetilde{\mathbf{b}}_m)$. We overload our definition of $\mathsf{E}(\cdot)$ in (4.6) and let $\mathsf{E}(\widetilde{\mathbf{b}}) \triangleq \sum_m \mathsf{E}(\widetilde{\mathbf{b}}_m) \in [0, M \log_2 K]$. Accordingly, we refer to the average of these losses over a training batch $\mathcal{T}$ as the *mean entropy*, given by

$$\frac{1}{TM} \sum_{i \in \mathcal{T}} \sum_{m=1}^{M} \mathsf{E}(\widetilde{\mathbf{b}}_m^{(i)}) = \frac{1}{TM} \sum_{i \in \mathcal{T}} \mathsf{E}(\widetilde{\mathbf{b}}^{(i)}). \tag{4.8}$$

In practice, introducing this loss will result in vectors $\widetilde{\mathbf{b}}$ that are only approximately binary, and hence, at test time, we project each block $\widetilde{\mathbf{b}}_m$ onto $\mathcal{K}_K$ using (4.5).

Uniform block support. Besides having blocks $\widetilde{\mathbf{b}}_m$ that resemble one-hot vectors, we would like for the supports of the binarized version $\mathbf{b}_m$ of $\widetilde{\mathbf{b}}_m$ to be as close to uniformly distributed as possible. This property allows the system to better exploit the support of our $\widetilde{\mathbf{b}}$ in encoding semantic information. It further contributes to the regularization of the model and encourages a better use of the available bit-rate.

We note first that one can estimate the distribution of the support of the $\mathbf{b}_m$ from a batch $\mathcal{T}$ using $\frac{1}{T} \sum_{i \in \mathcal{T}} \mathbf{b}_m^{(i)}$. Relaxing $\mathbf{b}_m$ to $\widetilde{\mathbf{b}}_m$ for training purposes, we want the entropy of this quantity to be high. This leads us to the definition of the negative *batch entropy* loss:

$$-\frac{1}{M} \sum_{m=1}^{M} \mathsf{E}\left(\frac{1}{T} \sum_{i \in \mathcal{T}} \widetilde{\mathbf{b}}_m^{(i)}\right) = -\frac{1}{M} \mathsf{E}\left(\overline{\mathbf{b}}\right), \tag{4.9}$$

where we let $\overline{\mathbf{b}} \triangleq \frac{1}{T} \sum_{i \in \mathcal{T}} \widetilde{\mathbf{b}}^{(i)}$.

Our learning objective (computed over a mini-batch) will hence be a standard classification objective further penalized by the mean and batch entropies in (4.8) and (4.9):

$$\begin{aligned} \mathsf{Loss}\big(\{(\mathbf{I}^{(i)}, y^{(i)})\}_{i \in \mathcal{T}}\big) \triangleq \frac{1}{T} \sum_{i \in \mathcal{T}} \Big[\ell(\mathbf{s}^{(i)}, y^{(i)}) + \\ \frac{\gamma}{M \log_2 K} \mathsf{E}(\widetilde{\mathbf{b}}^{(i)}) - \frac{\mu}{M \log_2 K} \mathsf{E}(\overline{\mathbf{b}}) \Big], \end{aligned} \tag{4.10}$$

with network output $\mathbf{s}$ defined as in (4.1), C the number of classes, and $\gamma > 0$ and $\mu > 0$ two hyper-parameters.

In our work, we use the following scaled version of the commonly used cross-entropy loss for classification:

$$\ell(\mathbf{s}, y) \triangleq -\frac{1}{\log_2 C} \log_2 \mathbf{s}[y]. \tag{4.11}$$

The scaling by $\log_2 C$ reduces the dependence of the hyper-parameters μ and γ on the number of classes C.

The training loss (4.10) is minimized with mini-batch stochastic gradient descent. The whole architecture can be learned this way, including the CNN feature extractor, the encoding layer and the classification layer (Fig. 4.1). Alternatively, (some of) the weights of the base CNN f_1 can be fixed to pre-trained values. In Section 4.3, we will consider the following variants, depending on set-ups: Training of FC_0/FC_1 only ("2-layer" training), the base CNN staying fixed; Training of $FC_{-1}/FC_0/FC_1$ ("3-layer training"); Training of all layers, $C_1 \cdots C_L$ and $FC_{-Q} \cdots FC_1$ ("full training").

4.2.3 Image search

As we will establish in Section 4.3, SUBIC yields important advantages in three image search applications, which we now describe along with a search complexity analysis.

Category and instance retrieval. These two tasks consist of ranking database images according to their similarity to a given query image, where similarity is defined by membership in a given semantic category *(category retrieval)* or by the presence of a specific object or scene *(instance retrieval)*. For these two tasks, we wish to use our structured binary representations to efficiently compute similarity scores for all database of images $\{\mathbf{I}^{(j)}\}_j$ given a query image $\mathbf{I}^*$. We propose using an asymmetric approach [66] that limits its query-side coding approximation: The database images are represented using their structured binary representation $\mathbf{b}^{(j)} = [\mathbf{b}_1^{(j)}; \cdots; \mathbf{b}_M^{(j)}] \in \mathcal{K}_K^M$, whereas the query image $\mathbf{I}^*$ is represented using the real-valued vector $\mathbf{z}^* = [\mathbf{z}_1^*; \cdots; \mathbf{z}_M^*] \in \mathbb{R}_+^{KM}$. Accordingly, the database images are ranked using the similarity score $(\mathbf{z}^*)^\top \mathbf{b}^{(j)}$. This expression also reads

$$\sum_{m=1}^{M} \mathbf{z}_m^* \big[\operatorname*{argmax}_r \mathbf{b}_m^{(j)}[r] \big], \tag{4.12}$$

which shows that M additions are needed to compute SUBIC similarities.

Image classification. A second important application is that of image classification in the case where the classes of interest are not known beforehand or change across time, as is the case of on-the-fly image classification from text queries [23, 22]. Having feature representations that are compact yet discriminative is important in this scenario, and a common approach to achieve this is to compress the feature vectors using PQ [20, 22]. The approach we propose is to instead use our supervised features to compactly represent the database images directly. New classes are assumed to be provided in the

form of annotated sets $\{(\mathbf{I}^{(q)}, y^{(q)})\}_q$ containing examples of the C' previously-unknown query classes,[3] and we learn classifiers from the structured codes $\widetilde{\mathbf{b}}$ of these examples. At test time, classifying a test feature $\mathbf{b}^{(j)} \in \mathcal{K}_K^M$ from the original dataset will require computing products $(\mathbf{W}^*)^\top \mathbf{b}^{(j)}$ (with $\mathbf{W}^* \in \mathbb{R}^{KM \times C'}$ for a softmax classifier or $\mathbf{W}^* \in \mathbb{R}^{KM}$ for a one-vs-rest classifier). Similarly to (4.12), this operation will likewise require only M additions per column of $\mathbf{W}^*$.

Search complexity relative to deep hashing. The expression (4.12) is reminiscent of the efficient distance computation mechanisms based on look-up-tables commonly used in product quantization search methods [66]. In particular, the expression in (4.12) establishes that computing the similarity between $\mathbf{z}_m^*$ and $\mathbf{b}_m$ incurs a complexity of M additions. This can be compared to the complexity incurred when computing the Hamming distance between two deep hash codes (*cf.* §4.1) $\mathbf{h}_1$ and $\mathbf{h}_2$ of length $B = M \log_2 K$ (*i.e.*, of storage footprint B equal to that of SUBIC): 1 XOR operation followed by as many additions as there are different bits in $\mathbf{h}_1$ and $\mathbf{h}_2$, a value that can be estimated from the expectation ($[\![\cdot]\!]$ is the Iverson bracket)

$$\mathbb{E}_{\mathbf{h}_1,\mathbf{h}_2}\left(\sum_{k=1}^{B} [\![\mathbf{h}_1[k] \neq \mathbf{h}_2[k]]\!]\right) = \frac{B}{2} = \frac{M}{2}\log_2 K, \tag{4.13}$$

if assuming i.i.d. and uniform $\mathbf{h}_j[k]$.

We note that $\mathcal{O}(1)$ look-up-table (LUT) based implementations of the Hamming distance are indeed possible, but only for small B (the required LUT size is 2^B). Alternatively, a smaller LUT of size $2^{B/M'}$ can be used by splitting the code into M' blocks (with M' comparable to M), resulting in a complexity $\mathcal{O}(M')$ comparable to the $\mathcal{O}(M)$ complexity of SUBIC.

4.3 Experiments

We assess the merits of the proposed supervised structured binary encoding for instance and semantic image retrieval by example and for database image classification, the three tasks described in Section 4.2.3.

Single-domain category retrieval. Single-domain category retrieval is the main experimental benchmark in the supervised binary hashing literature. Following the experimental protocol of [82], we report mean average precision (mAP) performance on the Cifar-10 database [25] which has 10 categories and 60k images of size 32×32 for each. The training is done on the 50k image training set. The test set is split into 9k database images and 1k query images, 100 per class. For fairness of comparison, we also use as

[3] Obtained from an external image search engine in on-the-fly scenarios.

Method	12-bit	24-bit	36-bit	48-bit
CNNH+ [129]	0.5425	0.5604	0.5640	0.5574
DLBHC [80]	0.5503	0.5803	0.5778	0.5885
DNNH [78]	0.5708	0.5875	0.5899	0.5904
DSH [82]	0.6157	0.6512	0.6607	0.6755
KSH-CNN [84]	-	0.4298	-	0.4577
DSRH [137]	-	0.6108	-	0.6177
DRSCH [134]	-	0.6219	-	0.6305
BDNN [31]	-	0.6521	-	0.6653
SUBIC (ours)	**0.6349**	**0.6719**	**0.6823**	**0.6863**

Table 4.2: Single-domain category retrieval. Comparison against published mAP values on Cifar-10 for various supervised deep hashing methods. See the *ImageNet* column of Table 4.3 for single-domain results on ImageNet.

Method	Pascal VOC	Caltech-101	ImageNet
PQ [66]	0.4965	0.3089	0.1650
CKM [99]	0.4995	0.3179	0.1737
LSQ [94]	0.4993	0.3372	0.1882
DSH-64 [82]	0.4914	0.2852	0.1665
SUBIC 2-layer	**0.5600**	0.3923	0.2543
SUBIC 3-layer	0.5588	**0.4033**	**0.2810**

Table 4.3: Cross-domain category retrieval. Performance (mAP) using 64-bit encoders across three different datasets using VGG-128 as base feature extractor. For completeness, results on ImageNet validation set (*i.e.* single-domain retrieval) are provided in the third column.

base CNN the same as introduced in [82]. It is composed of $L = 3$ convolutional layers with 32, 32 and 64 filters of size 5×5 respectively, followed by a fully connected layer FC_{-1} with $d = 500$ nodes. As proposed, we append to it a randomly initialized *encoder layer* FC_0 along with the classification layer FC_1. We fixed $K = 64$ and varied $M = \{2, 4, 6, 8\}$ so that $B = M \log_2(K)$ is equal to the desired bit-rate. Full training of the network is conducted, and hyper-parameters γ and μ are cross-validated as discussed later. We compare in Table 4.2 with various methods based on the same base CNN (top four rows, DSH [82], DNNH [78], DLBHC [80] and CNNH+ [129]), as well as other published values. For reference, we include a method (KSH-CNN [84]) not based on neural hash functions but using activations of a deep CNN as input features. Note that, at all bit-rates, from 12 to 48 bits, SUBIC outperforms these state-of-the-art supervised hashing techniques.

Cross-domain category retrieval. Using VGG-D with 128-D bottleneck (VGG-128) [21] as base CNN ($L = 5$, $Q = 3$ and $d = 128$), setting μ and γ to 1.0, we performed 2-layer and 3-layer learning of our network (see Section 4.2.2) on ILSVRC-ImageNet [56] training set. Two-layer training is conducted on 5k batches of $T = 200$ images. Three-layer training is initialized by previous one and run for another 5k batches. To evaluate cross-

Method	Oxford5K	Paris6K
PQ [66]	0.2374	0.3597
LSQ [94]	0.2512	0.3764
DSH-64 [82]	0.2108	0.3287
SUBIC	**0.2626**	**0.4116**

Table 4.4: Instance retrieval. Performance (mAP) comparison using 64-bit codes for all methods.

domain performance, we used our trained network to do category retrieval on Pascal Pascal VOC[120], Caltech-101 [17] and ImageNet validation sets. For each experiment, we used 1000 (2000 for ImageNet) random query images, the rest serving as database. The performance of the two trained SUBIC networks is reported in Table 4.3 at 64-bit rate ($M = 8$, $K = 256$). They are compared to three unsupervised quantization baselines, PQ [66], Cartesian k-means (CKM) [99] and LSQ [94], operating at 64-bit rate on VGG-128 image features. Further, to compare with supervised deep hashing approaches we implemented DSH [82] with VGG-128 as the base CNN, using their proposed loss and pair-wise training.

The impact of the proposed semantic supervision across domain is clearly demonstrated. Comparing unsupervised methods with our "2-layer" trained variant (no tuning of FC_1 is particularly enlightening since they all share exactly the same 128-dimensional input features). Training this representation as well in the "3-layer" version did not prove useful except on the ImageNet validation set. Note that the performance on this set could have been further improved through longer training, but at the expense on reduced transferability.

Instance retrieval. Unsupervised structured quantizers produce compact codes that enjoy state-of-the-art performance in instance retrieval at low memory footprint. Hence, in Table 4.4 we compare SUBIC to various such quantizers as well as DSH, using 64-bit representations for all methods. We used the *clean train* subset [47] of the Landmarks dataset [9] to train both DSH and a 2-layer SUBIC (the same as in Table 4.3, but with 60K batches). We report mAP on the Oxford5K[103] and Paris6K[104] datasets using their provided query/database split. SUBIC outperforms all methods while DSH performance is weaker to even unsupervised quantizers.

Image classification. In Table 4.5, we show how the 64-bit SUBIC encoding of VGG-128 features from Table 4.3 (2-layer variant) outperforms two baseline encoders with the same bit-rate for classification of compressed representations. As done in [22] for on-the-fly classification, the first baseline employs PQ [66] to represent the features compactly, and the second substitutes PQ by the better-performing CKM encoder [99]. Both unsupervised encoders are learned on VGG-128 features from the ImageNet training set. For the test on ImageNet, the two baselines employ the off-the-shelf VGG-128 classification layer as a classifier, reconstructing the PQ and CKM encoded versions beforehand (for

	ImageNet		Pascal VOC
	Top-1 acc.	*Top-5 acc.*	*mAP*
VGG-128*	53.80	77.32	73.79
PQ 64-bit	39.88	67.22	65.94
CKM 64-bit	41.15	69.66	67.25
SUBIC soft*	50.07	74.11	70.20
SUBIC 64-bit	47.77	72.16	67.86

Table 4.5: Classification performance with different compact codes. The rows marked (*) are non-binary codes. See the text for details.

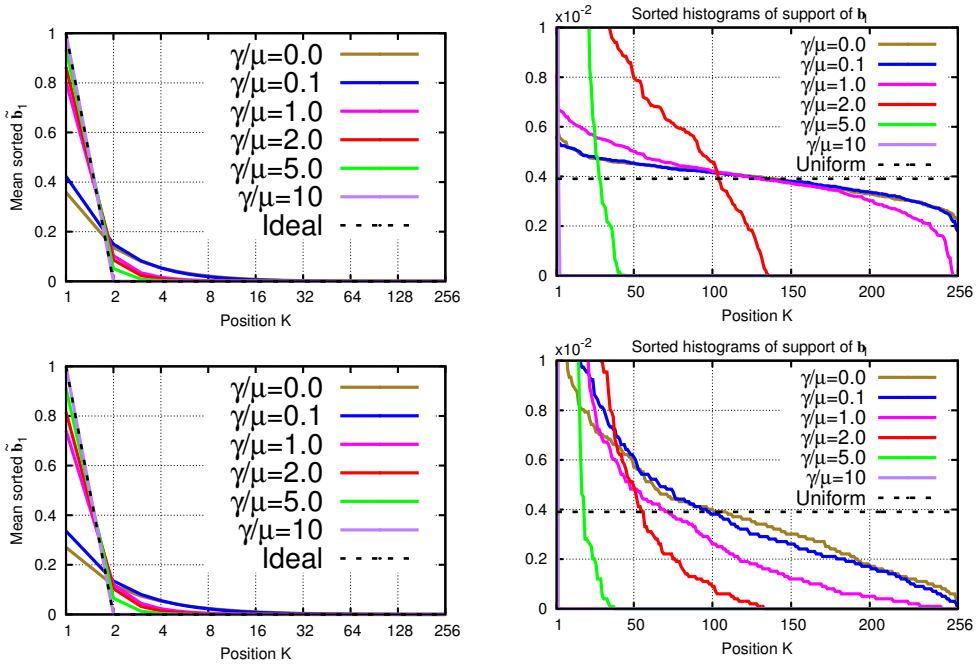


Figure 4.2: Effect of the entropy-based losses on the behavior of structured encoding. (*left*) One-hot encoding closeness of $\widetilde{\mathbf{b}}_1$. (*right*) Distribution of block support of $\mathbf{b}_1$. The black dashed curves correspond to the ideal, desired behavior. (*top*) ImageNet validation. (*bottom*) Pascal VOC.

reference, first row is for this classifier using original, un-coded features). Our results (bottom two rows) employ trained FC_1 layer applied to either $\widetilde{\mathbf{b}}$ code ("SUBIC soft") or $\mathbf{b}$ binary code ("SUBIC 64-bit"). In case of Pascal VOC we trained one-vs-rest SVM classifiers on the off-the-shelf VGG-128 features (top three rows) or on the $\widetilde{\mathbf{b}}$ features for SUBIC (bottom two rows).

Note that our compact SUBIC 64-bit features outperform both PQ and CKM features for the same bit-rate. Also notice that, although the classifiers for Pascal VOC are trained on block-softmax encoded features, when we use SUBIC 64-bit features the accuracy drops only marginally.

Structuring effectiveness of entropy-based losses. In Fig. 4.2 we evaluate our proposed entropy losses using the same SUBIC setup as in Table 4.5. We report statistics

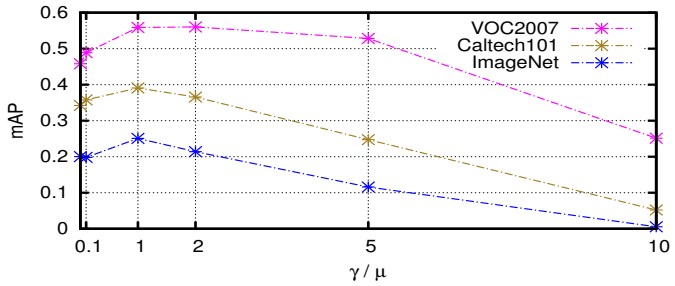


Figure 4.3: Effect of γ and μ on category retrieval performance.

on the ImageNet validation set (top graphs) and on all of Pascal VOC (bottom graphs), using the γ/μ ratio in the legends for ease of comparison.

To explore how well our γ-weighted mean-entropy loss favors codes resembling one-hot vectors, we extract the first 256-dimensional block $\widetilde{\mathbf{b}}_1$ from each image of the set (the seven other blocks exhibit similar behavior), re-order the entries of each such $\widetilde{\mathbf{b}}_1$ in decreasing order and average the resulting collection of vectors. The entries of this average vector are visualized for various values of γ/μ in the plots on the left. Ideal one-hot behavior corresponds to $[1; 0 \cdots ; 0]^\top$. On both datasets, increasing the mean entropy penalization weight γ relative to μ (*i.e.*, increasing γ/μ) results in code blocks that more closely resemble one-hot vectors.

To evaluate how well our μ-weighted negative batch-entropy term promotes uniformity of the support of the binarized blocks in $\mathbf{b}$, we plot, in the right side of Fig. 4.2, the sorted histograms of the support of the first block $\mathbf{b}_1$ over the considered image set. Note that increasing the weight μ of the batch entropy term relative to γ (decreasing γ/μ) results in distributions that are closer to uniform. As expected, the effect is more pronounced on the ImageNet dataset (top row) used as a training set, but extrapolates well to an independent dataset (bottom row).

We note further that it is possible (green curves, $\gamma/\mu = 5$) to have blocks that closely resemble one-hot vectors (left plot) but make poor use of the available support (0-valued histogram after the 47-th bin, on the right). It is likewise possible (blue curve, $\gamma/\mu = 0.1$) to enjoy good support usage with blocks that do not resemble one-hot vectors, establishing that our two losses work together to achieve the desired design goals.

Cross-validation of hyper parameters. Using the same setup and γ/μ values as in Fig. 4.2, in Fig. 4.3 we plot mAP as a function of γ/μ on three datasets. Note that the optimal performance (at $\gamma/\mu = 1$) for this architecture is obtained for an operating point that makes better use (closer to uniform) of the support of the blocks, as exemplified by the corresponding curves (pink) on the right in Fig. 4.2. This supports one of our original motivations that fostering uniformity of the support would encourage the system to use the support to encode semantic information.

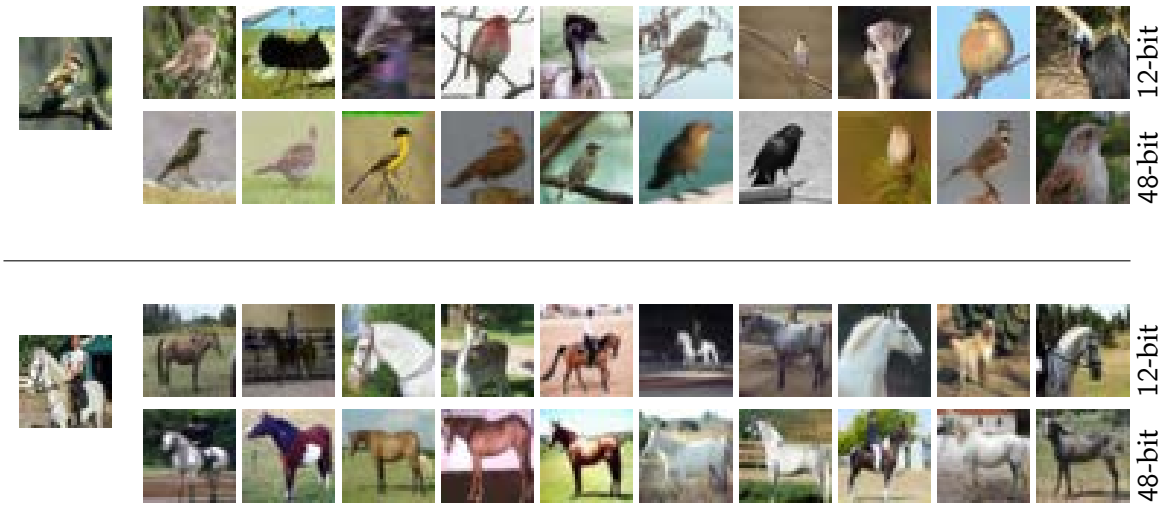


Figure 4.4: Category retrieval examples. Top ten ranked images retrieved from Cifar-10 for the query on the left when using 12-bit (*top*) and 48-bit (*bottom*) SUBIC. Note that higher bit-rates make the representation more sensitive to the query's orientation.

Category retrieval search examples. In Fig. 4.4 we present a search example when using the 12-bit and 48-bit SUBIC from Table 4.2. Note that increasing the bit rate results in retrieved images that are of the same pose as the query, suggesting that our method has potential for weakly-supervised (automatic) category refinement.

4.4 Conclusion

In this chapter we introduced SUBIC, a supervised, structured binary code produced by a simple encoding layer compatible with recent deep pipelines. Unlike previous deep binary hash codes, SUBIC features are block-structured, with each block containing a single active bit. We learn our proposed features in a supervised manner by means of a block-wise softmax non-linearity along with two entropy-based penalties. These penalties promote the one-hot quality of the blocks, while encouraging the active bits to employ the available support uniformly. While enjoying comparable complexity at fixed bit-rate, SUBIC outperforms the state-of-the art deep hashing methods in the single-domain category retrieval task, as well as state-of-the art structured vector quantizers in the instance retrieval task. SUBIC also outperforms structured vector quantizers in cross-domain category retrieval. Our method further showed promise for weakly-supervised semantic learning, a possible future direction.

CHAPTER

FIVE

LEARNING A COMPLETE IMAGE INDEXING PIPELINE

This chapter extends the work of Chapter 4 by learning a complete indexing pipeline which includes supervised learning of an inverted index and of an encoder. This chapter is based on the following publication:

Learning a complete image indexing pipeline. H. Jain, J. Zepeda, P. Pérez, and R. Gribonval. In Proceedings of the IEEE Conference on *Computer Vision and Pattern Recognition* (CVPR), 2018.

Decades of research have produced powerful means to extract features from images, effectively casting the visual comparison problem into one of distance computations in abstract spaces. Whether engineered, such as the bag of visual words representation [114] and its numerous variants [103, 104, 67, 138], or trained using convolutional deep networks, [83, 23, 22, 2, 47, 9], such vector representations are at the core of all content-based visual search engines. This applies particularly to example-based image retrieval systems where a query image is used to scan a database for images that are similar to the query in some way: in that they are the same image but one has been edited (*near duplicate detection*), or because they are images of the same object or scene (*instance retrieval*), or because they depict objects or scenes from the same semantic class (*category retrieval*).

Deploying such a visual search system requires conducting nearest neighbour search in a high-dimensional feature space. Both the dimension of this space and the size of the database can be very large, which imposes severe constraints if the system is to be practical in terms of storage (memory footprint of database items) and of computation (search complexity). Exhaustive exact search must be replaced by *approximate, non-exhaustive* search. To this end, two main complementary methods have emerged, both relying on variants of unsupervised vector quantization (VQ). The first such method, introduced by Sivic and Zisserman [114] is the inverted file system. Inverted files rely on a partitioning of the feature space into a set of mutually exclusive bins. Searching in a database thus amounts to first assigning the query image to one or several such bins, and then ranking the resulting shortlist of images associated to these bins using the Euclidean distance (or

some other distance or similarity measure) in feature space.

The second method, introduced by Jegou *et al*. [66] and as described in Chapter 3 and 4, consists of using efficient approximate distance computations as part of the ranking process. This is enabled by feature encoders producing compact representations of the feature vectors that further do not need to be decompressed when computing the approximate distances. This type of approaches, which can be seen as employing block-structured binary representations, superseded the (unstructured) binary hashing schemes that dominated approximate search.

Despite its impressive impact on the design of image representations [47, 2, 45, 106], supervised deep learning is still limited in what concerns the approximate search system itself. Most recent efforts focus on supervised deep binary hashing schemes, as discussed in the next section. As an exception, our work presented in Chapter 4 employs a block-structured approach inspired by the successful compact encoders referenced above. Yet the binning mechanisms that enable the usage of inverted files, and hence large-scale search, have not been addressed so far.

In this work we introduce a novel supervised inverted file system along with a supervised, block-structured encoder that together specify a complete, supervised, image indexing pipeline. Our design is inspired by the two methods of successful indexing pipelines described above, while extending ideas from our work in Chapter 4 to implement this philosophy.

Our main contributions are as follows:

(1) We propose the first, to our knowledge, image indexing system to reap the benefits of deep learning for both data partitioning and feature encoding.

(2) Our data partitioning scheme, in particular, is the first to replace unsupervised VQ by a supervised approach.

(3) We take steps towards learning the feature encoder and inverted file binning mechanism simultaneously as part of the same learning objective.

(4) We establish a wide margin of improvement over the existing baselines employing state-of-the-art deep features, feature encoders and binning mechanism.

5.1 Background

Approximating distances through compact encoding Concerning approximate distance computations, two main approaches exist. Hashing methods [125], on the one hand, employ Hamming distances between binary hash codes. Originally unsupervised, these methods have recently benefited from progress in deep learning

[129, 134, 137, 78, 80, 82, 31], leading to better systems for category retrieval in particular. Structured variants of VQ, on the other hand, produce fine-grain approximations of the high-dimensional features themselves through very compact codes [5, 39, 40, 60, 66, 73, 99, 135] that enable look-up table-based efficient distance computations. Contrary to recent hashing methods, VQ-based approaches have not benefited from supervision except in our proposed supervised deep learning approach SUBIC (Chapter 4), which leverages the advantages of structured compact encoding and yields state-of-the-art results on several retrieval tasks. In this chapter, we extend this supervised approach towards a complete indexing pipeline, that is, a system that also includes an inverted file index.

Scanning shorter lists with inverted indexes For further efficiency, approximate search is further restricted to a well chosen fraction of the database. This pruning is carried out by means of an Inverted File (IVF), which relies on a partitioning of the feature space into Voronoi cells defined using K-means clustering [64, 4]. Two things should be noted: The method to build the inverted index is unsupervised and it is independent from the way subsequent distance approximations are conducted (*e.g.*, while VQ is used to build the index, short lists can be scanned using binary embeddings [64]). In this work, we propose a unifying supervised framework. Both the inverted index and the encoding of features are designed and trained together for improved performance. In the next section, we expose in more detail the existing tools to design IVF/approximate search pipelines, before moving to our proposal in Section 5.3.

5.2 Review of image indexing

Image indexing systems are based on two main components: (i) an *inverted file* and *(ii)* a *feature encoder*. In this section we describe how these two main components are used in image indexing systems, thus laying out the motivation for the method we introduce in Section 5.3.

Inverted File (IVF) An inverted file relies on a partition of the database into mutually exclusive bins, a subset of which is searched at query time. The partitioning is implemented by means of VQ [114, 66, 4]: Given a vector $\mathbf{x} \in \mathbb{R}^d$ and a codebook $\mathbf{D} = [\mathbf{d}_k \in \mathbb{R}^d]_{k=1}^N$, the VQ representation of $\mathbf{x}$ in $\mathbf{D}$ is obtained by solving

$$n = \mathsf{argmin}_k \, \|\mathbf{x} - \mathbf{d}_k\|_2^2, \tag{5.1}$$

where n is the *codeword index* for $\mathbf{x}$ and $\mathbf{d}_n$ its *reconstruction*. Given a database $\{\mathbf{x}_i\}_i$ of image features, and letting n_i represent the codeword index of $\mathbf{x}_i$, the database is partitioned into N index bins $\mathcal{B}_n$. These bins, stored along with metadata that may include the features $\mathbf{x}_i$ or a compact representation thereof, is known as an inverted file. At query

time, the bins are ranked by decreasing pertinence $n_1, \dots, n_N$ relative to the query feature $\mathbf{x}^*$ so that

$$\|\mathbf{x}^* - \mathbf{d}_{n_1}\| \leq \dots \leq \|\mathbf{x}^* - \mathbf{d}_{n_N}\|, \tag{5.2}$$

i.e., by increasing order of reconstruction error. Using this sorting, one can specify a target number of images T to retrieve from the database and search only the first B bins so that $\sum_{k=1}^{B-1} |\mathcal{B}_{n_k}| \leq T \leq \sum_{k=1}^{B} |\mathcal{B}_{n_k}|$.

It is important to note that all existing state-of-the-art indexing methods employ a variant of the above described mechanism that relies on K-means-learned codebooks $\mathbf{D}$. To the best of our knowledge, ours is the first method to reap the benefits of deep learning to build an inverted file.

Feature encoder The inverted file outputs a shortlist of images with indices in $\bigcup_{k=1}^{B} \mathcal{B}_{n_k}$, which needs to be efficiently ranked in terms of distance to the query This is enabled by compact feature encoders that allow rapid distance computations without decompressing features. It is important to note that the storage bitrate of the encoding affects – besides storage cost – search speed, as higher bitrates means that bins need to be stored in secondary storage, where look-up speeds are a significant burden.

State-of-the art image indexing systems use feature encoders that employ a residual approach: A residual is computed from each database feature $\mathbf{x}$ and its reconstruction $\mathbf{d}_n$ obtained as part of the inverted file bin selection in (5.1):

$$\mathbf{r}_n = \mathbf{x} - \mathbf{d}_n. \tag{5.3}$$

This residual is then encoded using a very high resolution quantizer. Several schemes exist [24, 66] that exploit structured quantizers to enable low-complexity, high-resolution quantization, and herein we describe *product quantizers* and related variants [66, 99, 39]. Such vector quantizers employ a codebook $\mathbf{C} \in \mathbb{R}^{d \times K^M}$ with codewords that are themselves additions of codewords from M smaller *constituent* codebooks $\mathbf{C}_m = [\mathbf{c}_{m,k}]_k \in \mathbb{R}^{d \times K}$, $m = 1, \dots, M$, that are orthogonal ($\forall m \neq l, \mathbf{C}_m^{\mathrm{T}} \mathbf{C}_l = \mathbf{0}$):

$$\mathbf{C} = \Big[\sum\nolimits_{m=1}^{M} \mathbf{c}_{m,k_m} \Big]_{(k_1,\dots,k_M) \in (1,\dots,K)^M}. \tag{5.4}$$

Accordingly, an encoding of $\mathbf{r}$ in this structured codebook is specified by the indices $(k_1, \dots, k_M)$ which uniquely define the codeword $\mathbf{c}$ from $\mathbf{C}$, *i.e.*, the reconstruction of $\mathbf{r}$ in $\mathbf{C}$. Note that the bitrate of this encoding is $M \log_2(K)$.

Asymmetric distance computation Armed with such a representation for all database vectors, one can very efficiently compute an approximate distance between a query $\mathbf{x}^*$ and all database features $\mathbf{x} \in \{\mathbf{x}_i, i \in \cup_{k=1}^{B} \mathcal{B}_{n_k}\}$ in top-ranked bins. The residual of $\mathbf{x}^*$ for bin $\mathcal{B}_n$ is

$$\mathbf{r}_n^* = \mathbf{x}^* - \mathbf{d}_n \tag{5.5}$$

and the approach is asymmetrical in that this uncompressed residual is compared to the compressed, reconstructed residual representation $\mathbf{c}$ of the database vectors $\mathbf{x}$ in bin $\mathcal{B}_n$ using the distance

$$\|\mathbf{r}_n^* - \mathbf{c}\|_2^2 = \sum_{m=1}^{M} \|\mathbf{r}_n^* - \mathbf{c}_{m,k_m}\|_2^2. \tag{5.6}$$

We define the look-up tables (LUT)

$$\mathbf{z}_{n,m} \triangleq \left[\, \|\mathbf{r}_n^* - \mathbf{c}_{m,k}\|_2^2 \,\right]_k \in \mathbb{R}^K \tag{5.7}$$

containing the distances between $\mathbf{r}_n^*$ and all codewords of $\mathbf{C}_m$. Building these LUTs enables us to compute (5.6) using $\sum_{m=1}^{M} \mathbf{z}_m[k_m]$, an operation that requires only M table look-ups and additions, establishing the functional benefit of the encoding $(k_1, \dots, k_M)$.

To gain some insight into the above encoding, consider the one-hot representation $\mathbf{b}_m$ of the indices k_m given by

$$\mathbf{b}_m = \left[\left[\!\left[l = k_m \right]\!\right]\right]_l \in \mathcal{K}_K, \tag{5.8}$$

where $[\![\cdot]\!]$ denotes the Iverson brackets and we remind that:

$$\mathcal{K}_K \triangleq \{\mathbf{a} \in \{0,1\}^K, \|\mathbf{a}\|_1 = 1\}. \tag{5.9}$$

Using stacked column vectors

$$\mathbf{b} = [\mathbf{b}_1; \dots; \mathbf{b}_M] \in \mathcal{K}_K^M \text{ and} \tag{5.10}$$

$$\mathbf{z}_n = [\mathbf{z}_{n,1}; \dots; \mathbf{z}_{n,M}] \in \mathbb{R}_+^{MK}, \tag{5.11}$$

distance (5.6) can be expressed as follows:

$$\|\mathbf{r}_n^* - \mathbf{c}\|_2^2 = \mathbf{z}_n^{\mathrm{T}} \mathbf{b}. \tag{5.12}$$

Namely, computing approximate distances between a query $\mathbf{x}^*$ and the database features $\mathbf{x} \in \{\mathbf{x}_i, i \in \mathcal{B}_n\}$ amounts to computing an inner-product between a bin-dependent mapping $\mathbf{z}_n \in \mathbb{R}^{MK}$ of the query feature $\mathbf{x}^*$ and a block-structured binary code $\mathbf{b} \in \mathcal{K}_M^K$ derived from $\mathbf{x}$. A search then consists of computing all such approximate distances for the B most pertinent bins and then sorting the corresponding images in increasing order of these distances.

It is worth noting that most of the recent supervised binary encoding methods [129, 134, 137, 78, 80, 82, 31] do not use structured binary codes of the form $\mathbf{b}$ in (5.12). The main exception being SUBIC (Ch. 4), which further uses a sorting score that is an inner product of the same form as (5.12).

5.3 A complete indexing pipeline

The previous section established how state-of-the-art large-scale image search systems rely on two main components: an inverted file and a functional residual encoder that

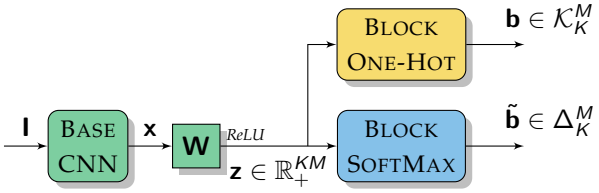


Figure 5.1: The SUBIC encoder from Chapter 4 operates on the feature vector $\mathbf{x}$ produced by a CNN to enable learning of (relaxed) block-structured codes ($\tilde{\mathbf{b}}$) $\mathbf{b}$. Blue, yellow, and green blocks are active, respectively, only at training time, only at testing time and at training/testing times.

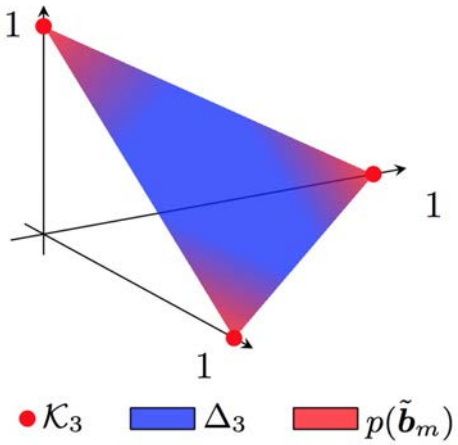


Figure 5.2: The discrete set $\mathcal{K}_3$ of one-hot encoded vectors, its convex-hull Δ_3, and the distribution of relaxed blocks $\tilde{\mathbf{b}}_m$ enforced by the SUBIC entropy losses. Omitting the negative batch entropy loss (5.19) would result in situations where $p(\tilde{\mathbf{b}}_m)$ is concentrated near only $k < 3$ of the elements in $\mathcal{K}_3$.

produces block-structured binary codes. While compact binary encoders based on deep learning have been explored in the literature, inverted file systems continue to rely on unsupervised K-means codebooks.

In this section we first revisit our SUBIC encoder, and then show how it can be used to implement a complete image indexing system that employs deep learning methodology both at the IVF stage and compact encoder stage.

5.3.1 Block-structured codes

The SUBIC encoder, which we recall in Fig. 5.1 for easy reference, is the first to leverage supervised deep learning to produce a block-structured code of the form $\mathbf{b} \in \mathcal{K}_K^M$ in (5.10). At learning time, the method relaxes the block-structured constraint. Letting

$$\Delta_K = \left\{ \mathbf{a} \in \mathbb{R}_+^K \text{ s.t. } \sum\nolimits_k \mathbf{a}[k] = 1 \right\} \tag{5.13}$$

denote the convex hull of $\mathcal{K}_K$, said relaxation

$$\tilde{\mathbf{b}} \in \Delta_K^M \tag{5.14}$$

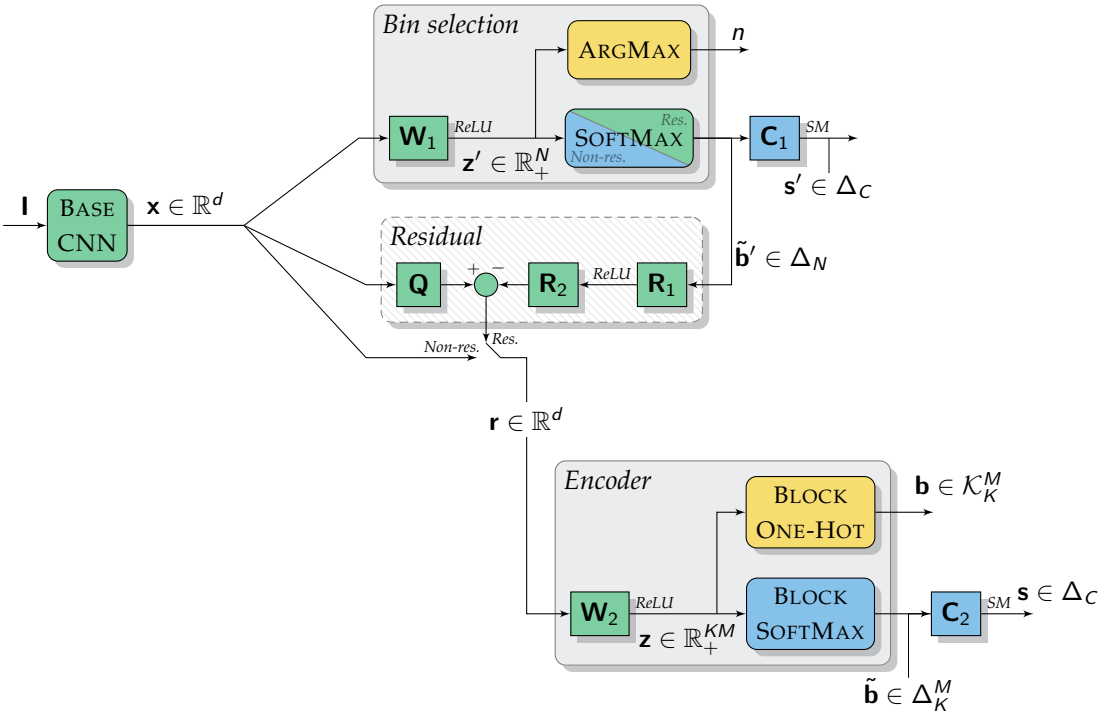


Figure 5.3: Proposed indexing architecture. The proposed indexing architecture consists of a bin selection component, a residual computation component, and a feature encoder. We use blocks with square corners (labeled with a weights matrix) to denote fully-connected linear operations, potentially followed by a ReLU or softmax (SM) nonlinearity. Blue, yellow, and green blocks are active, respectively, only at training time, only at testing (*i.e.* database indexing / querying) time and at training/testing times. The residual block can be disabled to define a new architecture, as illustrated by the switch at the bottom of the diagram.

is enforced by means of a fully-connected layer of output size KM and ReLU activation with output $\mathbf{z}$ that is fed to a *block softmax* non-linearity that operates as follows: Let $\mathbf{z}_m$ denote the m-th block of $\mathbf{z} \in \mathbb{R}^{KM}$ such that $\mathbf{z} = [\mathbf{z}_1; \dots; \mathbf{z}_M]$. Likewise, let $\tilde{\mathbf{b}}_m \in \Delta_K$ denote the m-th block of the relaxed code $\tilde{\mathbf{b}} \in \Delta_K^M$. The *block softmax* non-linearity operates by applying a standard softmax non-linearity to each block $\mathbf{z}_m$ of $\mathbf{z}$ to produce the corresponding block $\tilde{\mathbf{b}}_m$ of $\tilde{\mathbf{b}}$:

$$\tilde{\mathbf{b}}_m = \left[\frac{\exp\left(\mathbf{z}_m[k]\right)}{\sum_l \exp\left(\mathbf{z}_m[l]\right)} \right]_k . \tag{5.15}$$

At test time, the block-softmax non-linearity is replaced by a *block one-hot* encoder that projects $\tilde{\mathbf{b}}$ unto Δ_K^M. In practice, this can be accomplished by means of one-hot encoding of the index of the maximum entry of $\mathbf{z}_m$:

$$\mathbf{b}_m = \left[\, [\![k = \operatorname{argmax}(\mathbf{z}_m)]\!] \, \right]_k . \tag{5.16}$$

Our approach of SuBiC introduced two losses based on entropy that enforce the proximity of $\tilde{\mathbf{b}}$ to $\mathcal{K}_K^M$. The entropy of a vector $\mathbf{p} \in \Delta_K$, defined as

$$\mathrm{E}(\mathbf{p}) = \sum_{k=1}^{K} \mathbf{p}[k] \log_2 \left(\mathbf{p}[k]\right), \tag{5.17}$$

has a minimum equal to zero for deterministic distributions $\mathbf{p} \in \mathcal{K}_K$, motivating the use of the *entropy loss*

$$\ell_{\mathsf{E}}(\tilde{\mathbf{b}}) \triangleq \sum\nolimits_{m=1}^{M} \mathrm{E}(\tilde{\mathbf{b}}_m) \tag{5.18}$$

to enforce the proximity of the relaxed blocks $\tilde{\mathbf{b}}_m$ to $\mathcal{K}_K$. This loss on its own, however, could lead to situations where only some elements of $\mathcal{K}_K$ are favored (*cf*. Fig. 5.2), meaning that only a subset of the support of the $\mathbf{b}_m$ is used.

Yet entropy likewise has a maximum of $\log_2(K)$ for uniform distributions $\mathbf{p} = \frac{1}{K}\mathbf{1}$. This property can be used to encourage uniformity in the selection of elements of $\mathcal{K}_K$ by means of the *negative batch entropy loss*, computed for a batch $\mathcal{A} = \{\tilde{\mathbf{b}}^{(i)}\}_i$ of size $|\mathcal{A}|$ using

$$\ell_{\mathsf{B}}(\mathcal{A}) \triangleq -\sum\nolimits_{m=1}^{M} \mathrm{E}\Big(\frac{1}{|\mathcal{A}|}\sum_i \tilde{\mathbf{b}}_m^{(i)}\Big). \tag{5.19}$$

For convenience, we define the SUBIC loss computed on a batch $\mathcal{A}$ as the weighted combination of the two entropy losses, parametrized by the hyper-parameters $\gamma, \mu \in \mathbb{R}_+$:

$$\ell_{\mathsf{S}}^{\gamma,\mu}(\mathcal{A}) \triangleq \frac{\gamma}{|\mathcal{A}|}\sum\nolimits_{\tilde{\mathbf{b}}\in\mathcal{A}} \ell_{\mathsf{E}}(\tilde{\mathbf{b}}) + \mu\ell_{\mathsf{B}}(\mathcal{A}). \tag{5.20}$$

It is important to point out that, unlike the residual encoder described in Section 5.2, the SUBIC approach operates on the feature vector $\mathbf{x}$ directly. Indeed, the SUBIC method is only a feature encoder, and does not implement an entire indexing framework.

5.3.2 A novel indexing pipeline

We now introduce our proposed network architecture that uses the method of SUBIC described above to build an entire image indexing system. The system we design implements the main ideas of the state-of-the-art pipeline described in Section 5.2.

Our proposed network architecture is illustrated in Fig. 5.3. The input to the network is the feature vector $\mathbf{x}$ consisting of activation coefficients obtained by running a given image $\mathbf{I}$ through a CNN feature extractor. We refer to this feature extractor as the *base CNN* of our system.

Similarly to the design philosophy described in Section 5.2, our indexing system employs an IVF and a residual feature encoder. Accordingly, the architecture in Fig. 5.3 consists of two main blocks, *Bin selection* and *Encoder*, along with a *Residual* block that links these two main components.

Bin selection The first block, labeled *Bin selection* in Fig. 5.1 can be seen as a SUBIC encoder employing a single block (*i.e.* $M = 1$) of size N, with the block one-hot encoder substituted by an argmax operation. The block consists of a single fully-connected layer

with weight matrix $\mathbf{W}_1$ and ReLU activation followed by a second activation using softmax. When indexing a database image $\mathbf{I}$, this block is responsible for choosing the bin $\mathcal{B}_n$ that $\mathbf{I}$ is assigned to, using the argmax of the coefficients $\mathbf{z}'$.

Given a query image $\mathbf{I}^*$, the same binning block is responsible for sorting the bins in decreasing order of pertinence $\mathcal{B}_{n_1} \cdots \mathcal{B}_{n_N}$ using the coefficients $\mathbf{z}'^* \in \mathbb{R}_+^N$ so that

$$\mathbf{z}'^*[n_1] \geq \ldots \geq \mathbf{z}'^*[n_N], \tag{5.21}$$

in a manner analogous to (5.2).

(Residual) feature encoding Inspired by the residual encoding approach described in Section 5.2, we consider a block analogous to the residual computation of (5.3) and (5.5). The approach consists of building a vector (denoting ReLU as σ)

$$\mathbf{R}_2\sigma(\mathbf{R}_1\tilde{\mathbf{b}}'), \tag{5.22}$$

that is analogous to the reconstruction $\mathbf{d}_n$ of $\mathbf{x}$ obtained from the encoding n following the IVF stage (*cf*. (5.1) and discussion thereof), and subtracts it from a linear mapping of $\mathbf{x}$:

$$\mathbf{r} = \mathbf{Q}\mathbf{x} - \mathbf{R}_2\sigma(\mathbf{R}_1\tilde{\mathbf{b}}') \tag{5.23}$$

where R_1, R_2 and Q are learnable weight matrices. Besides the analogy to indexing pipelines, one other motivation for the above approach is to provide information to the subsequent feature encoding from the IVF bin selection stage (*i.e*. $\tilde{\mathbf{b}}'$) as well as the original feature $\mathbf{x}$. For completeness, as illustrated in Fig. 5.3, we also consider architectures that override this residual encoding block, setting $\mathbf{r} = \mathbf{x}$ directly.

The final stage consists of an M-block SUBIC encoder operating on $\mathbf{r}$ and producing test-time encodings $\mathbf{b} \in \mathcal{K}_K^M$, and training-time relaxed encoding $\tilde{\mathbf{b}} \in \Delta_K^M$. Note that, unlike the residual approach described in Section 5.2, our approach does not incurr the extra overhead required to compute LUTs using (5.7).

Searching Given a query image $\mathbf{I}^*$, it is first fed to the pipeline in Fig. 5.3 to obtain *(i)* the activation coefficients $\mathbf{z}'^*$ at the output of the $\mathbf{W}_1$ layer and *(ii)* the activation coefficients $\mathbf{z}^*$ at the output of the $\mathbf{W}_2$ layer. The IVF bins are then ranked as per (5.21) and all database images $\left\{\mathbf{I}_i,\ i \in \bigcup_{k=1}^{B} \mathcal{B}_{n_k}\right\}$ in the B most pertinent bins are sorted, based on their encoding $\mathbf{b}_i$, according to their score

$$\mathbf{z}'^{*\mathrm{T}}\mathbf{b}'_i + \mathbf{z}^{*\mathrm{T}}\mathbf{b}_i. \tag{5.24}$$

Training We assume we are given a training set $\{(\mathbf{I}^{(i)}, y^{(i)})\}_i$ organized into C classes, where label $y^{(i)} \in \{1, \ldots, C\}$ specifies the class of the i-th image. Various works on learning for retrieval have explored the benefit of using ranking losses like the triplet loss

and the pair-wise loss as opposed to the cross-entropy loss succesfully used in classification tasks [47, 129, 134, 137, 78, 80, 82, 31, 13]. Empirically, we have found that the cross-entropy loss yields good results in the retrieval task, and we adopt it in this work.

Given an image belonging to class c and a vector $\mathbf{p} \in \Delta_C$ that is an estimate of class membership probabilities, the cross-entropy loss is given by (the scaling is for convenience of hyper-parameter cross-validation)

$$\ell(\mathbf{p}, c) = -\frac{1}{\log_2 C} \log_2 \mathbf{p}[c]. \tag{5.25}$$

Accordingly, we train our network by enforcing that the relaxed block-structured codes $\tilde{\mathbf{b}}'$ and $\tilde{\mathbf{b}}$ are good feature vectors that can be used to predict class membership. We do so by feeding each vector to a soft-max classification layer (layers $\mathbf{C}_1$ and $\mathbf{C}_2$ in Fig. 5.3, respectively), thus producing estimates of class membership $\mathbf{s}'$ and $\mathbf{s}$ in Δ_C (*cf*. Fig. 5.3) from which we derive two possible task-related losses. Letting $\mathcal{T}$ denote a batch specified as a set of training-pair indices, these two losses are

$$L_{1,\alpha} = \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \left[\alpha \ell\big(\mathbf{s}'^{(i)}, y^{(i)}\big) + \ell\big(\mathbf{s}^{(i)}, y^{(i)}\big) \right] \tag{5.26}$$

$$\text{and } L_2 = \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \ell\big(\mathbf{s}'^{(i)} + \mathbf{s}^{(i)}, y^{(i)}\big), \tag{5.27}$$

where the scalar $\alpha \in \{0, 1\}$ is a selector variable. In order to enforce the proximity of the $\tilde{\mathbf{b}}'$ and $\tilde{\mathbf{b}}$ to $\mathcal{K}_N$ and $\mathcal{K}_K^M$, respectively, we further employ the loss

$$\Omega_{\mathcal{H}} = \ell_{\mathrm{S}}^{\gamma_1,\mu_1}\big(\{\tilde{\mathbf{b}}'^{(i)}\}_{i \in \mathcal{T}}\big) + \ell_{\mathrm{S}}^{\gamma_2,\mu_2}\big(\{\tilde{\mathbf{b}}^{(i)}\}_{i \in \mathcal{T}}\big), \tag{5.28}$$

which depends on the four hyper-parameters $\mathcal{H} = \{\gamma_1, \mu_1, \gamma_2, \mu_2\}$ (we disuss heuristics for their selection if §5.4).

Accordingly, the general learning objective for our system is

$$F_* = L_* + \Omega_{\mathcal{H}}, \tag{5.29}$$

and we consider three variants thereof:

(SUBIC-I) a non-residual variant with objective $F_{1,1}$ corresponding to independently training the bin selection block and the feature encoder;

(SUBIC-R) a residual variant with objective $F_{1,0}$ where the bin selection block is pre-trained and held fixed during learning; and

(SUBIC-J) a non-residual variant with objective F_2.

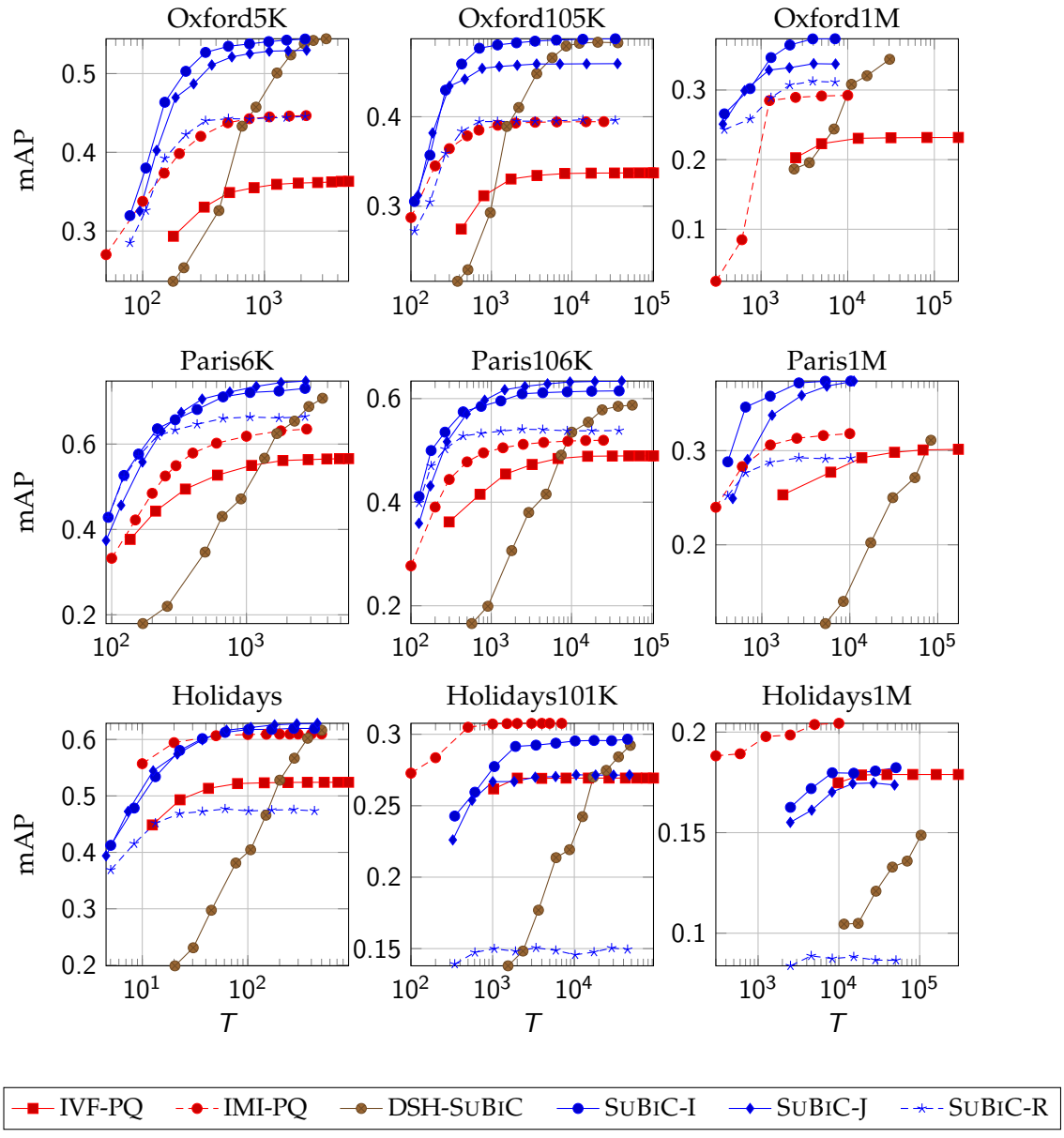


Figure 5.4: Large-scale image retrieval with complete pipelines. Plots of mAP vs. (average) shortlist size T. For all methods except IMI-PQ, the n-th plotted point is obtained from all images in the first $B = 2^n$ bins. For IMI-PQ, the mAP is computed on the first T responses.

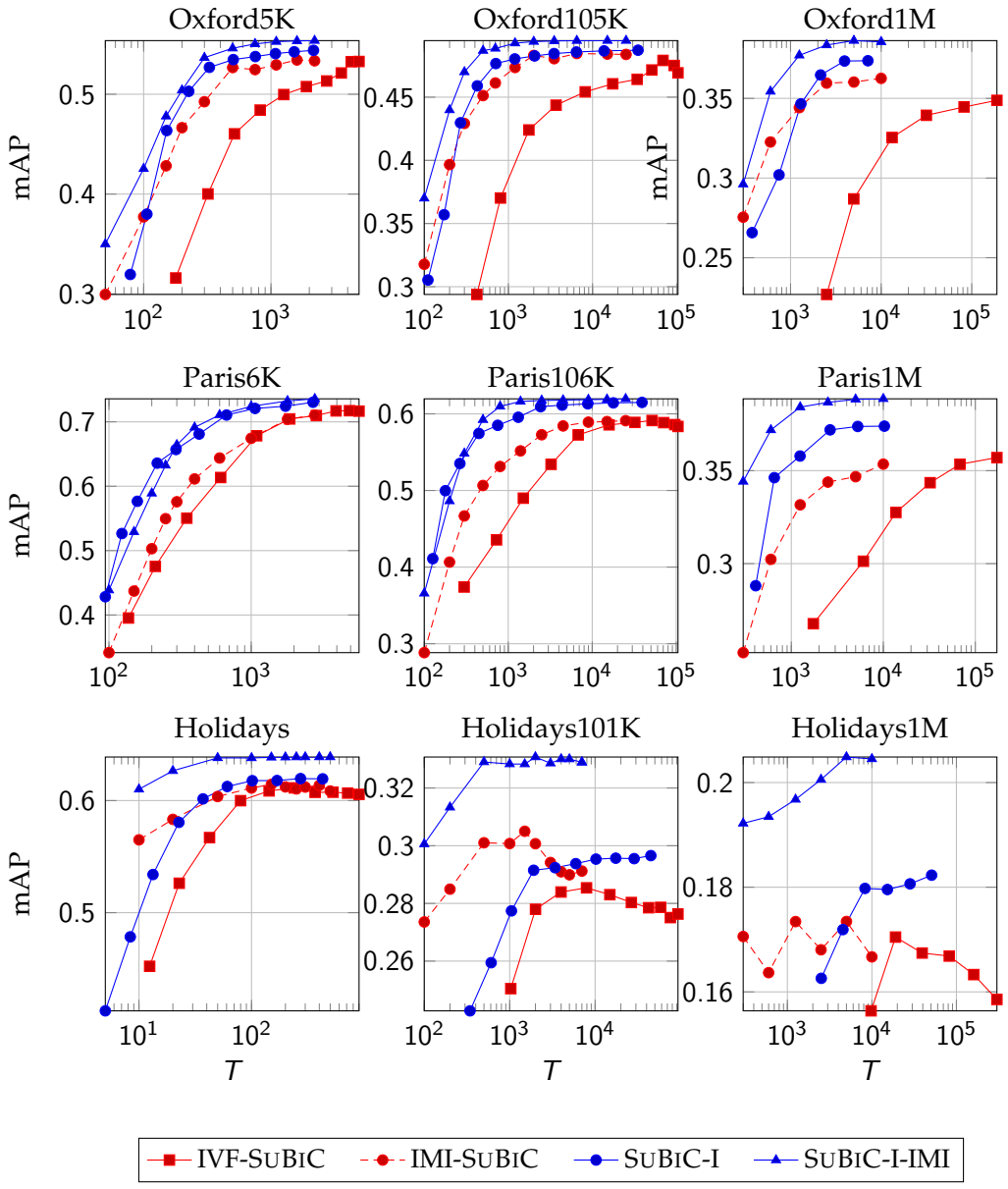


Figure 5.5: IMI variant of our approach and comparison for fixed encoder. Comparison of an IMI variant of our method to the baselines, when using the same (non-residual) feature encoder. Note the substantial relative improvements of SUBIC-I-IMI.

Method	Oxford5K	Oxford5K*	Paris6K	Holidays	Oxford105K	Paris106K
DIR [47]	84.94	84.09	93.58	90.32	83.52	89.10
PQ [66]	46.57	39.45	57.57	48.23	38.73	42.23
SUBIC [62]	**53.25**	**46.06**	**71.28**	**60.52**	**46.88**	**58.27**

Table 5.1: Instance retrieval with encoded features. Performance (mAP) comparison using 64-bit codes, first row shows reference results with original uncompressed features (8 Kbyte). When bounding box information is used for Oxford5K dataset, the performance degrades for both PQ and SUBIC, shown in column Oxford5K*, as both are trained on full images.

5.4 Experiments

Datasets For large-scale image retrieval, we use three publicly available datasets to evaluate our approach: Oxford5K [103][1], Paris6K [104][2] and Holidays [64][3]. For large-scale experiments, we add 100K and 1M images from Flickr (Flickr100K and Flickr1M respectively) as a noise set. For Oxford5K, bounding box information is not used. For Holidays, images are used without correcting orientation.

For training, we use the Landmarks-full subset of the Landmarks dataset [9][4], as in [47]. We could only get 125,610 images for the full set due to broken URLs. In all our experiments and for all approaches we use Landmarks-full as the training set.

For completeness, we also carry out *category retrieval* [62] test using the Pascal VOC[5] and Caltech-101[6] datasets. For this test, our method is trained on ImageNet.

Base features The base features $\mathbf{x}$ are obtained from the ResNet version of the network proposed in [47]. This network extends the ResNet-101 architecture with region of interest pooling, fully connected layers, and ℓ_2-normalizations to mimic the pipeline used for instance retrieval. Their method enjoys state-of-the-art performance for instance retrieval, motivating its usage as base CNN for this task.

Hyper-parameter selection For all three variants of our approach (SUBIC-I, SUBIC-J, and SUBIC-R), we use $N = 4096$ bins, and a SUBIC-$(8, 256)$ encoder having $M = 8$ blocks of $K = 256$ block size (corresponding to 8 bytes per encoded feature). These parameters correspond to commonly used values for indexing systems. To select the four hyper-parameters $\mathcal{H} = \{\gamma_1, \mu_1, \gamma_2, \mu_2\}$ in (5.29) we first cross-validate just the bin selection block to choose $\gamma_1 = 5.0$ and $\mu_1 = 6.0$. With these values fixed, we then cross-validate

[1] www.robots.ox.ac.uk/~vgg/data/oxbuildings/
[2] www.robots.ox.ac.uk/~vgg/data/parisbuildings/
[3] lear.inrialpes.fr/~jegou/data.php
[4] sites.skoltech.ru/compvision/projects/neuralcodes/
[5] http://host.robots.ox.ac.uk/pascal/VOC/
[6] http://www.vision.caltech.edu/Image_Datasets/Caltech101/

the encoder block to obtain $\gamma_2 = 0.6$ and $\mu_2 = 0.9$. We use the same values for all three variants of our system.

Evaluation of feature encoder First of all, we evaluate how SUBIC encoding performs on all the test datasets compared to the PQ unsupervised vector quantizer. We use $M = 8$ and $K = 256$ setups for both codes. SUBIC is trained for 500K batches of 200 training images, with $\gamma = 0.6$ and $\mu = 0.9$. The results reported in Table 5.1 show that, as expected, SUBIC outperforms PQ, justifying its selection as a feature encoder in our system. For reference, the first row in the table gives the performance with uncompressed features. While high, each base feature vector has a storage footprint of 8 Kilo bytes (assuming 4-byte floating points). SUBIC and PQ, on the other hand, require only 8 bytes of storage per feature (1000$\times$ less).

Baseline indexing systems We compare all three variants of our proposed indexing system against two existing baselines, as well as a straightforward attempt to use deep hashing as an IVF system:

(IVF-PQ) This approach uses an inverted file with $N = 4096$ bins followed by a residual PQ encoder with $M = 8$ blocks and constituent codebooks of size $K = 256$ (*cf*. (5.4)), resulting in an 8-byte feature size. The search employs asymmetric distance computation. During retrieval, the top $B = 2^n$, lists are retrieved, and, for each $n = 1, 2, \ldots$ the average mAP and average aggregate bin size T are plotted.

(IMI-PQ) The Inverted Multi-Index (IMI) [4] extends the standard IVF by substituting a product quantizer with $M = 2$ and $K = 4096$ in place of the vector quantizer. The resulting IVF has more than 16 million bins, meaning that, for practical testing sets (containing close to 1 million images), most of the bins are empty. Hence, when employing IMI, we select shortlist sizes T for which to compute average mAP to create our plots. Note that, given the small size of the IMI bins, the computation of the look-up tables $\mathbf{z}_n$ (*cf*. (5.7)) represents a higher cost per-image for IMI than for IVF. Furthermore, the fragmented memory reads required can have a large impact on speed relative to the contiguous reads implicit in the larger IVF bins.

(DSH-SUBIC) In order to explore possible approaches to include supervision in the IVF stage of an indexing system, we further considered using the DSH deep hash code [82] as a bin selector, carefully selecting the regularization parameter to be 0.03 by means of cross-validation. We train this network to produce 12-bit image representations corresponding to $N = 4096$ IVF bins, where each bin has an associated hash code. Images are indexed using their DSH hash, and at query time, the Hamming distance between the query's 12-bit code and each bin's code is used to rank the lists. For the encoder part, we used SUBIC with $M = 8$ and $K = 256$, the same used in Tab. 5.1.

Large-scale indexing Fig 5.4 shows the mAP performance versus average number of

retrieved images T for all three variants as well as the baselines described above. Note that the number of retrieved images is a measure of complexity, as for IVF, the time complexity of the system is dominated by the approximate distance computations in (5.12). For IMI, on the other hand, there is a non-negligible overhead on top of the approximate distance computation related to the large number of bins, as discussed above.

We present results for three datastets (Oxford5K, Paris6K, Holidays), on three different database scales (the original dataset, and when also including noise datasets of $100K$ and $1M$ images). Note that on Oxford5K and Paris6K, both SUBIC-I and SUBIC-J enjoy large advantages relative to all three baselines – at $T = 300$, the relative advantage of SUBIC-I over the IMI-PQ is 19% at least. SUBIC-R likewise enjoys an advangate on the Paris6K dataset, and performs comparably to the baselines on Oxford5K.

On Holidays SUBIC-I outperforms IVF-PQ by a large margin (18% relative), but does not outperform IMI-PQ. As discussed above, this comparison does not reflect the overhead implicit in an IMI index. To illustrate this overhead, we note that, when 1M images are indexed, the average (non-empty) bin size for IMI is 18.3, meaning that approximately 54.64 memory accesses and look-up table constructions need to be carried out for each IMI query per 1K images. This compares to an average bin size of 244.14 for IVF, and accordingly 4.1 contiguous memory reads and look-up table constructions. Note, on the other hand, that SUBIC-I readily outperforms IVF-PQ in all Holidays experiments.

Concerning the poor performance of SUBIC-R on Holidays, we believe this is due to poor generalization ability of the system because of the three extra fully-connected layers.

IMI extension Given the high performance of IMI for the Holidays experiments in Fig. 5.4, we further consider an IMI variant of our SUBIC-I architecture. To implement this approach, we learn a SUBIC-$(2, 4096)$ encoder (with $\gamma = 4$ and $\mu = 5$). Letting $\mathbf{z}'_m$ denote the m-th block of $\mathbf{z}'$, the $(k, l) \in \{1, \dots, 4096\}^2$ bins of SUBIC-IMI are sorted based on the score $\mathbf{z}'_1[k] + \mathbf{z}'_2[l]$. For fairness of comparison, we use the same SUBIC$-(8, 256)$ feature encoder for all methods including the baselines, which are IVF and IMI with unsupervised codebooks (all methods are non-residual). The results, plotted in Fig. 5.5, establish that, for the same number of bins, our method can readily outperform the baseline IMI (and IVF) methods. Furthermore, given that we use the best performing feature encoder (SUBIC) for all methods, this experiment also establishes that the SUBIC based binning system that we propose outperforms the unsupervised IVF and IMI baselines.

Category retrieval For completeness, we also carry out experiments in the category retrieval task which has been the main focus of recent deep hashing methods [129, 134, 137, 78, 80, 82, 31]. In this task, a given query image is used to rank all database images, with a correct match occuring for database images of the same class as the query image. For category retrieval experiments, we use VGG-M-128 base features [113], which have established good performance for classification tasks, and a SUBIC-$(1, 8192)$ for bin

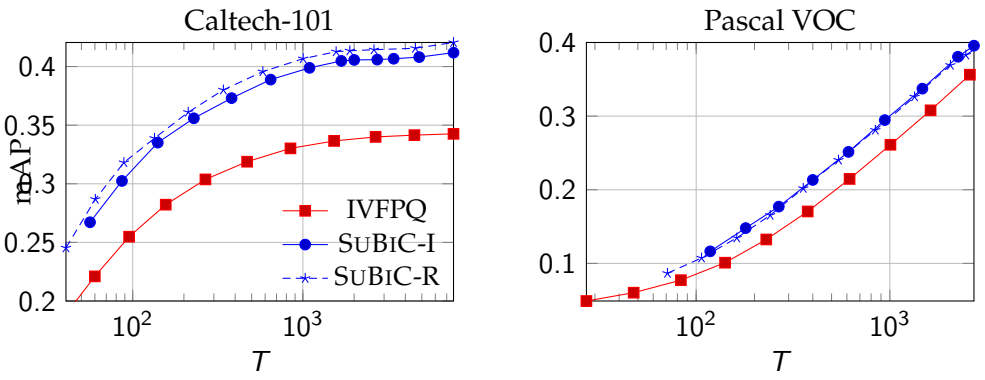


Figure 5.6: Category retrieval. Comparing SUBIC-I and SUBIC-R to IVF-PQ on the category retrieval task. joint-residual to non-joint SUBIC and IVFPQ for category retrieval on Pascal and Caltech101. All methods are trained on *VGG-M-128* features of ImageNet images.

selection. We use the ImageNet training set (1M+ images) to train, and the test (training) subsets of Pascal VOC and Caltech-101 as a query (respectively, database) set. We present results for this task in Fig. 5.6. Note that, unlike the Holidays experiments in Fig. 5.4, SUBIC-R performs best on Caltech-101 and equally well to SUBIC-I on Pascal VOC, a consequence of the greater size and diversity of the ImageNet datasets relative to the Landmarks dataset.

5.5 Conclusion

We present a full image indexing pipeline that exploits supervised deep learning methods to build an inverted file as well as a compact feature encoder. Previous methods have either employed unsupervised inverted file mechanisms, or employed supervision only to derive feature encoders. We establish experimentally that our method achieves state of the art results in large scale image retrieval.

CHAPTER

SIX

CONCLUSION AND PERSPECTIVES

6.1 Contributions

In this thesis, we address the problem of large scale image search. The large scale search is typically dealt by binary hashing or quantization based ANN search approaches. In this thesis, encouraged by the higher accuracy compared to hashing, we focus on quantization based approaches and propose various advances and enhancements to it. Typically, *structured vector quantization* methods for ANN search are based on unsupervised clustering by minimizing the reconstruction error and represent the data by sum of codewords. In one of our contributions, we propose an extension with weighted sums of the codewords for structured VQ. In two other major contributions, we explore supervised learning beyond reconstruction error minimization which leads to strong performance boost.

Next, we summarize our contributions and conclusion presented in the previous Chapters.

Weighted sum representation for VQ. Structured vector quantization uses a representation as a sum of codewords to encode the database points as discussed in Section 3.1, the objective is to compress the database points. In Chapter 3, we extend the VQ approaches by taking inspiration from sparse coding and propose a representation as weighted sum of codewords. The proposed representation is richer and lowers the reconstruction error. However, for this extension to be practical for large scale search, the newly introduced weights must be compressed. We propose to encode these weights by quantizing the vectors they form.

We apply our proposed representation to extend product quantization (PQ) and residual vector quantization (RVQ). Particularly in the case of RVQ, our proposed extension Qα-RVQ leads to strong improvements. This is due to the fact that we utilize the least informative bits in RVQ for encoding the scalar weights which encompasses significantly more information. While in case of PQ, our extension Qα-PQ provides a trade-off between accuracy and learning/encoding complexity. Our extension is complementary to

other extensions like optimized PQ (OPQ) as shown in experiments with Qα-OPQ. Further, our experiments on the billion-size BIGANN dataset showed that, combined with inverted indexing systems like IVF or IMI, our extension improves accuracy with similar search times.

Supervised structured binary representation. Structured VQ approaches for large scale search benefit from *asymmetric distance computation* (ADC) and, high representation capacity due its *structure* of multiple codebooks. But VQ could be limited by reconstruction error minimization and not taking advantage of labeled data. With the motivation to benefit from ADC with structured code as in VQ and supervised training as in deep hashing approaches, we propose SUBIC in Chapter 4. In this work, we introduce a trainable layer on top of a CNN, that encodes images into structured binary codes which are block wise one-hot (*i.e.*, concatenation of binary vectors having all entries but one being zero). To enforce the structure we propose two entropy-based losses, *mean entropy loss* to induce a one-hot block structure, and *negative batch entropy loss* to encourage uniformity in the code. These entropy-based losses are combined with a cross-entropy loss to train the CNN in a supervised manner for image classification.

Our proposed approach of SUBIC outperformed the state-of-the-art methods in single-domain and cross-domain image search with compressed representation. The compact codes produced by our approach are also useful for large scale classification with transferability to new datasets or classes, as shown in Section 4.3.

Supervised inverted indexing. As shown in Chapter 3 on the BIGANN dataset, inverted indexing is an essential part of large scale search. It is built by clustering the database points with VQ. In Chapter 5 we propose to apply our work of SUBIC to build inverted indexes in a supervised manner. Our approach led to strong improvements compared to the unsupervised baselines. It is the first approach to replace unsupervised VQ based inverted indexing with *supervised inverted indexing*.

Unifying the supervised indexing pipeline. In Chapter 5, we also propose a unified learning of the inverted index and the encoder. We explore three ways of unified learning including SUBIC-I which combines separately trained inverted index and encoder. All three provide large improvements on unsupervised VQ based indexing on most of our experiments.

Within our approaches, as observed in Chapter 5, in case of instance retrieval with limited training data (125K images), SUBIC-I outperformed the residual based SUBIC-R. While for category retrieval with large training on ImageNet (1.28M images), SUBIC-R surpasses SUBIC-I. This suggests that larger training data could further improve the performance of our jointly learned methods.

6.2 Perspectives

In this section, we discuss possible directions to advance this work and other research problems which might benefit from the work presented in this thesis.

6.2.1 Setting the hyper-parameters in SUBIC

In Chapter 4, we introduced two entropy-based losses which act together to induce a one-hot block structure with statistical uniformity in the position of the active bit. Figure 4.2 shows the importance of the balance between the hyper-parameters γ and μ which weight these losses in the cost function as in the Eq. (4.10). We find this balance by monitoring the entropies at the training time. If the values of these hyper-parameters are good, then the *mean entropy* (should be low) and *batch entropy* (should be high) will be as desired. Otherwise, both the entropies would be either low or high which shows the loss of uniformity or structure respectively as in Figure 4.2. Thus, ideally, we should be able to set or compute the values for γ and μ based on the entropies while training. That is, if the batch entropy is low then we must increase μ or if mean entropy is high then γ should be increased. Such a way to set γ and μ would make the SUBIC approach free from the need of heuristically finding the hyper-parameters. Also, it might possibly improve the performance as γ and μ can be updated throughout the training.

6.2.2 Training with limited annotated data and high bit-rate encoding

As we have seen in Chapters 4 and 5, supervised deep learning of compact codes has high potential and can achieve large improvements over VQ or binary hashing based methods. But these supervised approaches need large amount of labeled data for training, which might be expensive to get or may not be available. When training data is limited, learning a model with a large number of parameters over-fits and does not generalize well. This is a common problem with many supervised approaches. There are various ways to regularize the model to improve generalization [46, 121, 115, 58, 14] such as *data augmentation*, ℓ_2 *parameter regularization*, *Dropout*, *Batch normalization*, *DropConnect*. These approaches are certainly applicable to SUBIC also when trained end-to-end. In many of our experiments presented in Chapters 4 and 5 we have used a pre-trained base CNN (which benefits from the various regularizations to better generalize) on top of which we train our encoder layer. Now if the base CNN gives high-dimensional features, for example the features from DIR [47] are 2048 dimensional, and suppose that we use an encoding layer which produces MK dimensional structured binary code then, the encoding layer would have $2048 \times MK$ parameters. Now if we use high bit-rate i.e. large M and K, we have more parameters in the encoding layer which might cause over-fitting. In our experiments we have seen performance saturation when increasing the

bit-rate. We list some possible ways to handle it:

- PCA compression of the base CNN features or adding a layer before the encoding layer to project the features to lower dimensions.
- Using a partially connected encoding layer instead of the fully connected one.
- Training with a decoder to reconstruct the input vector for the structured binary code, which would allow the use of unlabeled data.

Even with the limited availability of annotated data, the ability to encode with higher bit-rate would improve performance significantly and would thus expand further the applicability of the proposed approach.

6.2.3 Attribute discovery

By recognizing the characteristic "attributes" of an object or of a class of objects, we can define this object or this class. Such a definition with attributes is useful for identifying new objects or classes. With our approach presented in Chapter 4, we learn an MK dimensional structured binary code which has exactly M active bits and the rest being zero. We think that many of these MK positions are representing some attributes and few of these attributes could be identifiable visual concepts such as eyes, legs, round or rectangular shapes, etc. It would be interesting to find those in some (semi-)automatic way as MK is too large to proceed manually. This could be useful for classifying unseen images with no training data, as in zero-shot classification [79], based on the attributes marked by the active bits.

6.2.4 Video retrieval

Retrieving videos from a large collection based on a query video requires addressing the same problems as with still images, namely indexing and encoding with compact codes. But video search bring new challenges as image sequences provide more diverse and complex visual information [55], while increasing dramatically the amount of redundancy, hence of ambiguity. The methodological contributions of this thesis being not restricted to still images, (or, in fact, to visual information), they might be useful to address the specific challenges of video search. This would require however new developments to handle at best the specifics of video search and constitutes as such an interesting research direction of its own.

PUBLICATIONS

International conferences

- Himalaya Jain, Patrick Pérez, Rémi Gribonval, Joaquin Zepeda, and Hervé Jégou
  Approximate search with quantized sparse representations
  In Proceedings of European Conference on Computer Vision (ECCV), 2016 [60].
- Himalaya Jain, Joaquin Zepeda, Patrick Pérez, and Rémi Gribonval
  SUBIC*: A supervised, structured binary code for image search*
  In Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017 [62].
- Himalaya Jain, Joaquin Zepeda, Patrick Pérez, and Rémi Gribonval
  Learning a complete image indexing pipeline
  In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018 [61].

ACRONYMS

ADC	Asymmetric Distance Computation
ANN	Approximate Nearest Neighbor
AQ	Additive Quantization
CKM	Cartesian K-means
CNN	Convolutional Neural Network
CQ	Composite Quantization
DSH	Deep Supervised Hashing
ERVQ	Enhanced Residual Vector Quantization
IMI	Inverted Multi-Index
IVF	Inverted File
LSH	Locality Sensitive Hashing
LUT	Look-Up Table
mAP	mean Average Precision
MSE	Mean Squared Error
NN	Nearest Neighbor
OPQ	Optimized Product Quantization
PQ	Product Quantization
Qα-PQ	Qα-Product Quantization
Qα-RVQ	Qα-Residual Vector Quantization
RVQ	Residual Vector Quantization
SDC	Symmetric Distance Computation
SCQ	Sparse Composite Quantization
VQ	Vector Quantization

LIST OF FIGURES

LIST OF TABLES

BIBLIOGRAPHY

[1] L. Ai, J. Yu, Z. Wu, Y. He, and T. Guan. Optimized residual vector quantization for efficient approximate nearest neighbor search. *Multimedia Systems*, pages 1–13, 2015.

[2] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. NetVLAD: CNN architecture for weakly supervised place recognition. In *Proc. Int. Conf. Computer Vision*, 2016.

[3] R. Arandjelovic and A. Zisserman. All about VLAD. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2013.

[4] A. Babenko and V. Lempitsky. The inverted multi-index. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2012.

[5] A. Babenko and V. Lempitsky. Additive quantization for extreme vector compression. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2014.

[6] A. Babenko and V. Lempitsky. The inverted multi-index. *IEEE Trans. Pattern Anal. Machine Intell.*, 37(6):1247–1260, 2015.

[7] A. Babenko and V. Lempitsky. Tree quantization for large-scale similarity search and classification. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2015.

[8] A. Babenko and V. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *Proc. Conf. Comp. Vision Pattern Rec.*, pages 2055–2063, 2016.

[9] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky. Neural codes for image retrieval. In *Proc. Europ. Conf. Computer Vision*, 2014.

[10] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Structured sparsity through convex optimization. *Statistical Science*, pages 450–468, 2012.

[11] C. F. Barnes, S. A. Rizvi, and N. M. Nasrabadi. Advances in residual vector quantization: A review. *IEEE Transactions on Image Processing*, 5(2):226–262, 1996.

[12] M. Bawa, T. Condie, and P. Ganesan. Lsh forest: self-tuning indexes for similarity search. In *Proceedings of the 14th international conference on World Wide Web*, pages 651–660. ACM, 2005.

[13] C. Bilen, J. Zepeda, and P. Perez. The CNN News Footage Datasets : Enabling Supervision in Image Retrieval. *EUSIPCO*, 2016.

[14] M. Blot, T. Robert, N. Thome, and M. Cord. Shade: Information-based regularization for deep learning. *arXiv preprint arXiv:1804.10988*, 2018.

[15] A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[16] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166, 1997.

[17] Caltech-101dataset. www.vision.caltech.edu/Image_Datasets/Caltech101/.

[18] Y. Cao, M. Long, J. Wang, H. Zhu, and Q. Wen. Deep quantization network for efficient image retrieval. In *Proc. AAAI Conf. on Artificial Intelligence*, 2016.

[19] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

[20] K. Chatfield, R. Arandjelović, O. Parkhi, and A. Zisserman. On-the-fly learning for visual search of large-scale image and video datasets. *Int. J. Multimedia Information Retrieval*, 4(2):75–93, 2015.

[21] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *Proc. British Machine Vision Conf.*, 2014.

[22] K. Chatfield, K. Simonyan, and A. Zisserman. Efficient on-the-fly category retrieval using ConvNets and GPUs. In *Proc. Asian Conf. Computer Vision*, 2014.

[23] K. Chatfield and A. Zisserman. Visor: Towards on-the-fly large-scale object category retrieval. In *Proc. Asian Conf. Computer Vision*, 2012.

[24] Y. Chen, T. Guan, and C. Wang. Approximate nearest neighbor search by residual vector quantization. *Sensors*, 10(12):11259–11273, 2010.

[25] Cifar-10 dataset. www.cs.toronto.edu/~kriz/cifar.html.

[26] T. M. Cover and J. A. Thomas. *Elements of information theory 2nd edition*. Wiley-interscience, 2006.

[27] Q. Dai, J. Li, J. Wang, and Y.-G. Jiang. Binary optimized hashing. In *Proc. ACM Int. Conf. Multimedia*, 2016.

[28] A. Dasgupta, R. Kumar, and T. Sarlós. Fast locality-sensitive hashing. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1073–1081. ACM, 2011.

[29] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *ASCG*, 2004.

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2009.

[31] T.-T. Do, A.-D. Doan, and N.-M. Cheung. Learning to hash with binary deep neural network. In *Proc. Europ. Conf. Computer Vision*, 2016.

[32] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li. Modeling lsh for performance tuning. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 669–678. ACM, 2008.

[33] M. Douze, H. Jégou, and F. Perronnin. Polysemous codes. In *Proc. Europ. Conf. Computer Vision*, pages 785–801. Springer, 2016.

[34] T. Durand, T. Mordan, N. Thome, and M. Cord. Wildcat: Weakly supervised learning of deep convnets for image classification, pointwise localization and segmentation. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2017.

[35] T. Durand, N. Thome, and M. Cord. Weldon: Weakly supervised learning of deep convolutional neural network. In *Proc. Conf. Comp. Vision Pattern Rec.*, pages 4743–4752, 2016.

[36] M. Elad. *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*. Springer, 2010.

[37] J. H. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of the American statistical Association*, 76(376):817–823, 1981.

[38] P. Frossard, P. Vandergheynst, R. F. i Ventura, and M. Kunt. A posteriori quantization of progressive matching pursuit streams. *IEEE Transactions on Signal Processing*, 52(2):525–535, 2004.

[39] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization for approximate nearest neighbor search. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2013.

[40] T. Ge, K. He, and J. Sun. Product sparse coding. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2014.

[41] A. Gersho and R. M. Gray. *Vector quantization and signal compression*, volume 159. Springer Science & Business Media, 2012.

[42] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. 25th Int. Conf. on Very Large Data Base, 1999*, 1999.

[43] H. Goh, N. Thome, M. Cord, and J.-H. Lim. Top-down regularization of deep belief networks. In *Advances in Neural Information Processing Systems*, pages 1878–1886, 2013.

[44] H. Goh, N. Thome, M. Cord, and J.-H. Lim. Learning deep hierarchical visual feature coding. *IEEE transactions on neural networks and learning systems*, 25(12):2212–2225, 2014.

[45] Y. Gong, L. Wang, R. Guo, and S. Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *Proc. Europ. Conf. Computer Vision*, 2014.

[46] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[47] A. Gordo, J. Almazán, J. Revaud, and D. Larlus. Deep image retrieval: Learning global representations for image search. In *Proc. Europ. Conf. Computer Vision*, 2016.

[48] D. Gorisse, M. Cord, and F. Precioso. Salsas: Sub-linear active learning strategy with approximate k-nn search. *Pattern Recognition*, 44(10-11):2343–2357, 2011.

[49] D. Gorisse, M. Cord, and F. Precioso. Locality-sensitive hashing for chi2 distance. *IEEE transactions on pattern analysis and machine intelligence*, 34(2):402–409, 2012.

[50] R. Gray. Vector quantization. *IEEE ASSP Magazine*, 1(2):4–29, 1984.

[51] R. M. Gray and D. L. Neuhoff. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383, 1998.

[52] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. Conf. Comp. Vision Pattern Rec.*, pages 770–778, 2016.

[53] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon. Spherical hashing. In *Proc. Conf. Comp. Vision Pattern Rec.*, pages 2957–2964. IEEE, 2012.

[54] J.-P. Heo, Z. Lin, and S.-E. Yoon. Distance encoded product quantization. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2014.

[55] W. Hu, N. Xie, L. Li, X. Zeng, and S. Maybank. A survey on visual content-based video indexing and retrieval. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(6):797–819, 2011.

[56] Ilsvrc-ImageNet dataset. `www.image-net.org/challenges/LSVRC/`.

[57] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[58] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[59] M. Iwamura, T. Sato, and K. Kise. What is the most efficientway to select nearest neighbor candidates for fast approximate nearest neighbor search? In *Proc. Int. Conf. Computer Vision*, pages 3535–3542. IEEE, 2013.

[60] H. Jain, P. Pérez, R. Gribonval, J. Zepeda, and H. Jégou. Approximate search with quantized sparse representations. In *Proc. Europ. Conf. Computer Vision*, 2016.

[61] H. Jain, J. Zepeda, P. Pérez, and R. Gribonval. Learning a complete image indexing pipeline. *arXiv preprint arXiv:1712.04480*, 2017.

[62] H. Jain, J. Zepeda, P. Pérez, and R. Gribonval. SuBiC: A supervised, structured binary code for image search. In *Proc. Int. Conf. Computer Vision*, 2017.

[63] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. In *Proc. Conf. Comp. Vision Pattern Rec.*, pages 1–8. IEEE, 2008.

[64] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. *Proc. Europ. Conf. Computer Vision*, 2008.

[65] H. Jégou, M. Douze, and C. Schmid. Searching with quantization: approximate nearest neighbor search using short codes and distance estimators. Technical report, Inria, 2009.

[66] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Machine Intell.*, 33(1):117–128, 2011.

[67] H. Jegou, F. Perronnin, M. Douze, J. Sánchez, P. Perez, and C. Schmid. Aggregating local image descriptors into compact codes. *IEEE Trans. Pattern Anal. Machine Intell.*, 34(9):1704–1716, 2012.

[68] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg. Searching in one billion vectors: Re-rank with source coding. In *Proc. Int. Conf. Acoust., Speech, Signal Processing*, 2011.

[69] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. Proximal Methods for Sparse Hierarchical Dictionary Learning. In *ICML*, 2010.

[70] J. Ji, J. Li, S. Yan, B. Zhang, and Q. Tian. Super-bit locality-sensitive hashing. In *Advances in Neural Information Processing Systems*, pages 108–116, 2012.

[71] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *arXiv:1702.08734*, 2017.

[72] B. H. Juang and A. J. Gray. Multiple stage vector quantization for speech coding. In *Proc. Int. Conf. Acoust., Speech, Signal Processing*, 1982.

[73] Y. Kalantidis and Y. Avrithis. Locally optimized product quantization for approximate nearest neighbor search. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2014.

[74] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Int. Conf. Neural Inf. Proc. Systems*, 2012.

[75] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2009.

[76] B. Kulis, P. Jain, and K. Grauman. Fast similarity search for learned metrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2143–2157, 2009.

[77] P. Kulkarni, F. Jurie, J. Zepeda, P. Pérez, and L. Chevallier. SPLeaP: Soft pooling of learned parts for image classification. In *Proc. Europ. Conf. Computer Vision*, 2016.

[78] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2015.

[79] C. H. Lampert, H. Nickisch, and S. Harmeling. Attribute-based classification for zero-shot visual object categorization. *IEEE Trans. Pattern Anal. Machine Intell.*, 36(3):453–465, 2014.

[80] K. Lin, H.-F. Yang, J.-H. Hsiao, and C.-S. Chen. Deep learning of binary hash codes for fast image retrieval. In *Conf. Comp. Vision Pattern Rec. Workshops*, 2015.

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

[82] H. Liu, R. Wang, S. Shan, and X. Chen. Deep supervised hashing for fast image retrieval. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2016.

[83] R. Liu, Y. Zhao, S. Wei, Z. Zhu, L. Liao, and S. Qiu. Indexing of CNN features for large scale image search. *arXiv preprint arXiv:1508.00217*, 2015.

[84] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2012.

[85] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1–8. Citeseer, 2011.

[86] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proc. Conf. Comp. Vision Pattern Rec.*, pages 3431–3440, 2015.

[87] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[88] Q. Lv, M. Charikar, and K. Li. Image similarity search with compact data structures. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 208–217. ACM, 2004.

[89] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*, pages 950–961. VLDB Endowment, 2007.

[90] X. Lyu, J. Zepeda, and P. Pérez. Maximum margin linear classifiers in unions of subspaces. In *Proc. British Machine Vision Conf.*, 2016.

[91] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(Jan):19–60, 2010.

[92] S. G. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on signal processing*, 41(12):3397–3415, 1993.

[93] G. S. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*, pages 141–150. ACM, 2007.

[94] J. Martinez, J. Clement, H. H. Hoos, and J. J. Little. Revisiting additive quantization. In *Proc. Europ. Conf. Computer Vision*, 2016.

[95] J. Martinez, H. H. Hoos, and J. J. Little. Stacked quantizers for compositional vector compression. *arXiv preprint arXiv:1411.2173*, 2014.

[96] Y. Matsui, Y. Uchida, H. Jégou, and S. Satoh. A survey of product quantization. *ITE Transactions on Media Technology and Applications*, 6(1):2–10, 2018.

[97] Y. Mu, J. Shen, and S. Yan. Weakly-supervised hashing in kernel space. In *Proc. Conf. Comp. Vision Pattern Rec.*, pages 3344–3351. IEEE, 2010.

[98] Y. Mu and S. Yan. Non-metric locality-sensitive hashing. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[99] M. Norouzi and D. Fleet. Cartesian k-means. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2013.

[100] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov. Hamming distance metric learning. In *Advances in neural information processing systems*, pages 1061–1069, 2012.

[101] M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in hamming space with multi-index hashing. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2012.

[102] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *Proc. Conf. Comp. Vision Pattern Rec.*, pages 1–8. IEEE, 2007.

[103] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2007.

[104] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2008.

[105] D. Picard, M. Cord, and A. Revel. Image retrieval over networks: Active learning using ant algorithm. *IEEE Transactions on Multimedia*, 10(7):1356–1365, 2008.

[106] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. In *Conf. Comp. Vision Pattern Rec. Workshops*, pages 512–519. IEEE, 2014.

[107] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[108] A. Sablayrolles, M. Douze, H. Jégou, and N. Usunier. How should we evaluate supervised hashing? *arXiv preprint arXiv:1609.06753*, 2016.

[109] V. Satuluri and S. Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *Proceedings of the VLDB Endowment*, 5(5):430–441, 2012.

[110] P. Schönemann et al. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966.

[111] F. Shen, C. Shen, W. Liu, and H. T. Shen. Supervised discrete hashing. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 7, pages 37–45. IEEE, 2015.

[112] F. Shen, C. Shen, Q. Shi, A. Van Den Hengel, and Z. Tang. Inductive hashing on manifolds. In *Proc. Conf. Comp. Vision Pattern Rec.*, pages 1562–1569. IEEE, 2013.

[113] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[114] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proc. Int. Conf. Computer Vision*, 2003.

[115] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[116] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proc. Conf. Comp. Vision Pattern Rec.*, June 2015.

[117] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proc. Conf. Comp. Vision Pattern Rec.*, pages 1701–1708, 2014.

[118] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *Proc. Conf. Comp. Vision Pattern Rec.*, pages 1–8. IEEE, 2008.

[119] A. Vedaldi and A. Zisserman. Sparse kernel approximations for efficient classification and detection. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2012.

[120] Pascal Pascal VOC dataset. `host.robots.ox.ac.uk/pascal/VOC/voc2007/`.

[121] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[122] D. Wang, C. Otto, and A. K. Jain. Face search at scale. *IEEE Trans. Pattern Anal. Machine Intell.*, 2016.

[123] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large-scale search. *IEEE Trans. Pattern Anal. Machine Intell.*, 34(12):2393–2406, 2012.

[124] J. Wang, W. Liu, S. Kumar, and S. Chang. Learning to hash for indexing big data - A survey. *CoRR*, 2015.

[125] J. Wang, H. T. Shen, J. Song, and J. Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014.

[126] N. Wang and D.-Y. Yeung. Learning a deep compact image representation for visual tracking. In *Advances in neural information processing systems*, pages 809–817, 2013.

[127] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in neural information processing systems*, pages 1753–1760, 2009.

[128] J. Wright, Y. Ma, J. Mairal, G. Sapiro, T. S. Huang, and S. Yan. Sparse representation for computer vision and pattern recognition. *Proceedings of the IEEE*, 98(6):1031–1044, 2010.

[129] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *Proc. AAAI Conf. on Artificial Intelligence*, 2014.

[130] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu. Complementary hashing for approximate nearest neighbor search. In *Proc. Int. Conf. Computer Vision*, 2011.

[131] M. Yaghoobi, T. Blumensath, and M. Davies. Quantized sparse approximation with iterative thresholding for audio coding. In *Proc. Int. Conf. Acoust., Speech, Signal Processing*, 2007.

[132] J. Zepeda, C. Guillemot, and E. Kijak. The iteration-tuned dictionary for sparse representations. In *IEEE Workshop on Multimedia Signal Processing*, 2010.

[133] J. Zepeda, C. Guillemot, and E. Kijak. Image compression using sparse representations and the iteration-tuned and aligned dictionary. *IEEE Journal of Selected Topics in Signal Processing*, 2011.

[134] R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang. Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *IEEE Trans. Image Processing*, 24(12):4766–4779, 2015.

[135] T. Zhang, C. Du, and J. Wang. Composite quantization for approximate nearest neighbor search. In *Proc. Int. Conf. Machine Learning*, 2014.

[136] T. Zhang, G.-J. Qi, J. Tang, and J. Wang. Sparse composite quantization. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2015.

[137] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *Proc. Conf. Comp. Vision Pattern Rec.*, 2015.

[138] X. Zhou, K. Yu, T. Zhang, and T. S. Huang. Image classification using super-vector coding of local image descriptors. In *Proc. Europ. Conf. Computer Vision*, 2010.