MEMORY-EFFICIENT IMAGE DATABASES FOR MOBILE VISUAL SEARCH

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

David M. Chen
March 2014

This dissertation is online at: http://purl.stanford.edu/fn367xx2931

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Bernd Girod, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Robert Gray

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Radek Grzeszczuk

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

Mobile visual search (MVS) systems compare query images captured by the mobile device's camera against a database of labeled images to recognize objects seen in the device's viewfinder. Practical MVS systems require a fast response to provide an interactive and compelling user experience. We show how a memory-efficient database stored on a mobile device can effectively enhance the capabilities of MVS systems and enable fast and accurate queries in different environments.

The image signatures stored in the database on the mobile device must be compact to fit in the device's small memory capacity, capable of fast comparisons across a large database, and robust against large visual distortions. We first develop two methods for efficiently compressing a database constructed from feature histograms. Our methods reduce the database memory usage by 4-5$\times$ without any loss in matching accuracy and possess fast decoding capabilities. Subsequently, we develop a third database representation based on feature residuals that is even more compact. Compressed residuals reduce the database memory usage by 12-14$\times$, require only a small codebook, and enable image matching directly in the compressed domain.

With a compact database stored on a mobile device, we implement an MVS system that can recognize media covers, book spines, outdoor landmarks, artwork, and video frames in less than 1 second per query. Our system uses robust motion analysis on the device to automatically infer user interest, select high-quality query frames, and update the pose of recognized objects for accurate augmentation. We also demonstrate how a continuous stream of compact image signatures enables a low bitrate query expansion onto a remote server. The query expansion improves image matching during the current query and updates the local on-device database to benefit future queries.

Acknowledgements

First and foremost, I would like to thank Prof. Bernd Girod, my thesis advisor. I feel very fortunate to have been a PhD student in his research group. His advice and guidance have played a pivotal role in shaping the direction of my research. Because he encouraged me to explore many interesting research problems, I gained a broad perspective on different areas and found interesting connections among these areas. He also advised me to pursue research that has a rigorous theoretical foundation but at the same time has clear practical applications, which is a great principle for me to carry into the next stage of my career. While serving as a teaching assistant for Prof. Girod's digital image processing class, I had the unique opportunities to develop new mobile image processing resources, homework assignments, examination problems, and lecture examples. These experiences gave me deeper insights into many image processing topics, which in turn benefited various aspects of my research.

Next, I would like to thank the other members of my thesis reading committee. I would like to thank Prof. Robert Gray for his helpful advice and suggestions on my research, especially the fundamentals of vector quantization, and for always being generous with his time in discussing my research progress. I would like to thank Dr. Radek Grzeszczuk for being a caring mentor of my research. When I worked at Nokia Research Center under Dr. Grzeszczuk's guidance, I learned a lot about how to build robust mobile applications and large-scale systems. Also, I would like to thank Prof. Tor Raubenheimer and Prof. Ayfer Ogzur for their helpful advice and suggestions during my oral exam.

I enjoyed many fruitful collaborations and discussions with the members of the Image, Video, and Multimedia Systems (IVMS) group. At the beginning, I benefited

from the advice and help of Aditya Mavlankar during a project on compressing the Laplacian pyramid. Then, I learned much about distributed video coding from David Varodayan and Markus Flierl. In the middle of my PhD program, I collaborated closely with a dream team of researchers – Sam Tsai, Vijay Chandrasekhar, Gabriel Takacs, Mina Makar, and Ngai-Man Cheung – on topics closely related to my thesis. I worked successfully with Derek Pang for several years on the digital image processing class. More recently, I have enjoyed working with Peter Vajda, Matt Yu, Huizhong Chen, Andre Araujo, and Maryam Daneshi on developing a personalized television news system, and I have benefited from many research discussions with Roland Angst. Kelly Yilmaz has also been incredibly generous in helping with many research and class related issues.

My research also has benefited from close interactions with external collaborators. I would like to thank the members of Radek Grzeszczuk's team – Ramakrishna Vedantham, Timo Pylvanainen, Jerome Berclaz, Vasu Parameswaran, and Varsha Hedau – for their collaboration on projects at Nokia Research Center including product recognition, landmark recognition, and road sign recognition. I would like to thank Jatinder Singh, Chenghsin Hsu, Kyu-Han Kim, and Rahul Swaminathan for their collaboration on projects at Deutsche Telekom R-and-D Labs including media cover recognition and asset tracking. I would like to thank Prof. Marc Pollefeys, Georges Baatz and Kevin Koeser at ETH Zurich and Georg Schroth at Technische Universitat Munchen for their collaboration on landmark recognition research. I would like to thank Prof. Shih-Fu Chang, Brendan Jou, Hongzhi Li, and Joe Ellis at Columbia University for their collaboration on the personalized television news project, as part of the Brown Institute for Media Innovation.

Finally, I would like to thank my parents for their support of my education. Their encouragement was helpful in enduring the ups and downs of my graduate studies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mobile visual search (MVS) refers to an emerging class of applications where a mobile device takes a photo of the real world, recognizes objects in the photo, and retrieves the corresponding information and metadata about these objects from an online database [79] or a local database [37]. Analogous to typing a text query into a search engine to retrieve desired information, MVS uses a photo as a visual query to search an image database. In recent years, mobile devices are becoming better equipped with faster CPUs and GPUs, larger memory capacity, higher resolution touch screen displays for content viewing and user input, and more accurate sensors including GPS transceivers, accelerometers, compasses, and gyroscopes. MVS researchers are taking advantage of these improved capabilities on mobile devices to engineer faster and more robust recognition systems. The left side of Figure 1.1 shows three examples of MVS systems that we have built for recognizing media covers, book spines, and outdoor landmarks. In each case, the user simply points the mobile device's camera at objects in the scene to retrieve useful information about these objects quickly and reliably from a database. The retrieved metadata is processed and overlaid on top of the recognized objects in the viewfinder. Recent commercial deployments of similar MVS applications include Google Goggles [82], Nokia Point-and-Find [160], Amazon Flow [6], and Kooaba Augmented Reality [115].

When recognizing objects like media covers, book spines, and outdoor landmarks, the MVS system utilizes a pre-constructed database of labeled images, like

Figure 1.1: Mobile visual search (MVS) systems for media cover recognition, book spine recognition, and outdoor landmark recognition. In each case, a query image is compared against a large database of labeled images to recognize objects that appear in the query image.

the database depicted on the right side of Figure 1.1. Large collections of annotated images are widely available today from commercial vendors, social media sites, and search engines. If the query image can be efficiently compared against a database to find the most visually similar database images, then the labels and metadata for the best matching database images can be retrieved for augmenting the viewfinder.

Matching visual queries against an image database, however, is a much harder problem than matching clearly typed text queries against a text database. As evident in the examples in Figure 1.1, the most visually similar database images often show the objects in different poses and illuminations than in the query image. To successfully match different images of the same objects, the computer vision community has designed and developed scale- and rotation-invariant local image features such as SIFT [133, 134], SURF [20, 18], CHoG [33, 34], RIFF [203, 204], and ORB

[181] which are highly robust to typical image distortions. These features can be effectively matched between different images of the same objects, even when there are severe geometric changes, lighting variations, object occlusions, and distracting clutter. Typically, several hundred local features are extracted for each image to robustly characterize the image, so a database of 1M images needs to efficiently represent at least several hundred million local features altogether.

To incorporate large-scale database comparisons between sets of local image features into a low-latency MVS system, three major technical challenges must be solved:

- The database must be preprocessed and organized into a representation that can be searched very quickly. For the interactive MVS applications that we are creating, it is important to be able to efficiently search a database on the order of 1M images in around 1 second or less.

- A large database consumes a substantial amount of memory. Since the database must be stored in random access memory (RAM) for fast access, the database's large memory footprint can cause performance bottlenecks on a server that is running other memory-intensive tasks concurrently or prevent the database from being stored on a mobile device that has only a small memory capacity.

- The database must consist of image signatures that are intrinsically robust against various visual distortions often encountered in MVS queries. Even though a query image and a database image will show an object from different vantage points and under different illumination conditions, we must be able to reliably identify that it is actually the same object in both images from the chosen image representations.

In this thesis, we address these three major technical challenges simultaneously. We design and develop image databases that are fast to search, light in memory usage, and robust against common geometric and photometric visual distortions. Because the databases are compact, these databases can be stored either on a server or on a mobile device. Our memory-efficient databases enable a variety of image matching architectures, including an on-device image matching system, an on-server image

matching system, and a hybrid image matching system that combines the advantages of on-device and on-server search. Hence, we can engineer dynamically adaptive MVS systems that are highly resilient against fluctuations in the network or server load to provide a fast and accurate response to the user on every new query.

1.1 Research Contributions

The major contributions of this thesis are described as follows:

- *Memory-efficient feature histograms*: We develop a compressed representation of feature histograms for large-scale image retrieval. Our compression framework can be applied to any of the multitude of retrieval systems based on the popular bag-of-words model [193, 158], which has been pervasive in the computer vision literature during the last decade. The compressed feature histograms require substantially less memory and can even yield slightly better retrieval accuracy than uncompressed feature histograms. We show how the compressed histograms can be decoded quickly during a query using word-aligned codecs. We also analyze the fundamental reasons behind the achieved memory savings.

- *Memory-efficient feature residuals*: We develop a compressed representation of feature residuals for large-scale image retrieval. The residual generation and comparison stages are carefully optimized, so that feature residuals generated with a small codebook can attain the same retrieval accuracy as feature histograms generated with a much larger codebook. Compact feature residuals can also be compared directly in the compressed domain, thus avoiding any decoding delay during a query. Using a multi-round search algorithm, the compressed residuals can be compared even more efficiently throughout a large database without an exhaustive search. We also analyze how retrieval accuracy varies with the codebook size and the number of local features per image for optimal selection of the image signature parameters in an MVS system.

- *Robust, low-latency MVS systems*: Using the memory-efficient image databases that we have created, we then build robust, low-latency MVS systems that can recognize a variety of objects. First, we develop an on-device image matching system that stores and searches a compact database entirely on the mobile device. On-device search guarantees low query latencies anywhere or anytime, without any dependence on external network or server conditions. Our system incorporates a motion-adaptive query selection mechanism that automatically infers user interest, selects high-quality query frames, and initiates queries only when the user is interested in the contents of the viewfinder. Subsequently, we develop a hybrid image matching system that combines the advantages of on-device and on-server image matching. An efficient query expansion onto a remote server and update of the local on-device database are achieved simultaneously, by transmitting a low-bitrate interframe-coded stream of compact signatures.

1.2 Organization

The remaining chapters in this thesis are organized as follows:

- In Chapter 2, we review the literature that is most relevant to our research. First, we describe the most popularly used local feature keypoint detectors and descriptors. Next, we show how local features extracted from pairs of images can be compared with methods like approximate nearest neighbor (ANN) matching and random sample consensus (RANSAC). Then, we describe methods for searching a large database of images based on feature histograms or feature residuals generated from the optimally selected local features which are extracted from images. Finally, we describe other low-latency MVS systems and explain how the MVS systems developed in this thesis differ from the other systems.

- In Chapter 3, we create a compression framework for generating memory-efficient feature histograms. First, we describe how to optimize the retrieval

accuracy of a vocabulary tree with greedy multi-path search and soft binning. Next, we develop tree histogram coding (THC) to directly compress the database tree histograms generated with a vocabulary tree. Then, we show how to further speed up the histogram comparisons with an inverted index. We develop inverted index coding (IIC) to substantially reduce the inverted index's memory footprint. Both THC and IIC are designed to have fast decoding capabilities by employing word-aligned codecs that minimize the number of memory accesses during a query. Finally, we evaluate the large-scale image retrieval performance of THC and IIC for different vocabulary tree sizes and show the benefits of these database compression methods.

- In Chapter 4, we create a framework for generating memory-efficient feature residuals. First, we build a compact, discriminative global image signature called the residual enhanced visual vector (REVV) through a sequence of key operations: nonlinear mapping by a power law, global linear transform, vector quantization, residual aggregation and normalization, cell-specific linear transform, and signed binarization. To optimally choose the codebook size and the number of local features for generating a REVV signature, we perform a statistical analysis of REVV's retrieval performance. Then, we show how the compact REVV signatures can be effectively compared in the compressed domain with weighted correlations that reward observations which are likely to originate from matching image pairs. Hence, weighted correlations effectively increase the separability of matching and non-matching images. We further speed up the database search with a multi-round scoring algorithm that quickly discards non-matching database images. Finally, we compare the large-scale image retrieval performance of REVV against THC and IIC from Chapter 3 and against another state-of-the-art global image signature, and we highlight the advantages of REVV over these other methods.

- In Chapter 5, we build robust, low-latency MVS systems for recognizing objects such as media covers, book spines, and outdoor landmarks. First, we create an on-device image matching system that has three major processing

stages: motion-adaptive query selection, local database search and information retrieval with REVV signatures, and viewfinder augmentation. Then, we develop a more general hybrid image matching system that primarily performs local database search, but occasionally queries a database hosted on a remote server. To minimize the transmission latency when contacting the remote server, we develop techniques for efficient interframe coding of a continuous stream of REVV signatures extracted from viewfinder frames. We verify mathematically and experimentally that the bitrate is substantially reduced by interframe coding compared to independent coding of REVV signatures.

- Finally, in Chapter 6, we summarize the main results achieved in this thesis, discuss important lessons learned from our research, and list some open questions for possible future work.

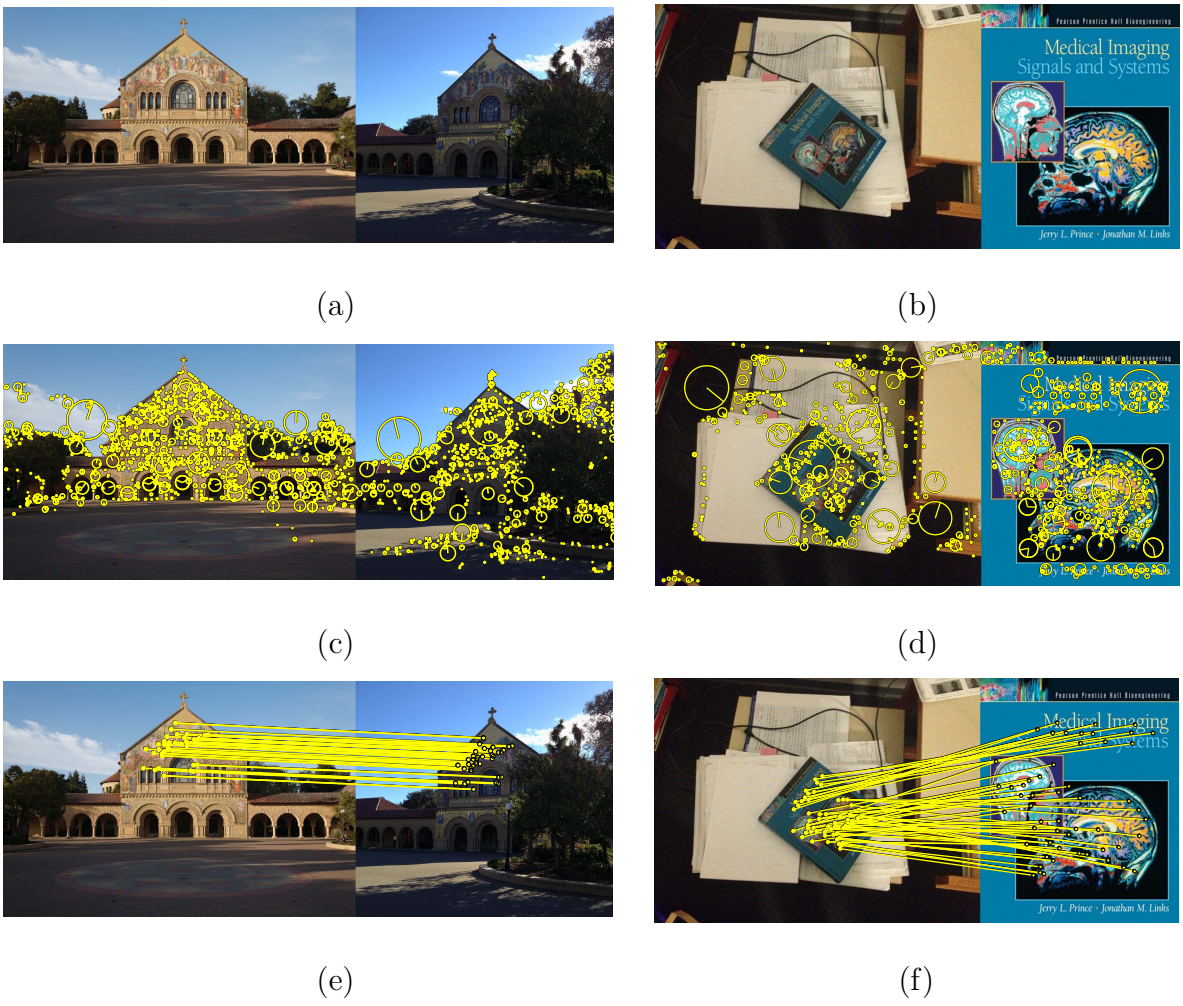
Chapter 2

Background

Most mobile visual search (MVS) systems require efficient and scalable methods for matching sets of images, so this chapter first reviews the important related work in robust image matching and retrieval. Section 2.1 describes methods for local feature extraction. Then, Section 2.2 explains how features extracted from two images can be reliably matched to discover the common object(s) in the images. Since the methods in Section 2.2 are too computationally intensive to apply to every image in a large database, Section 2.3 explains the process of forming and matching feature histograms and feature residuals for efficient large-scale image retrieval. Finally, Section 2.4 describes related mobile augmented reality (MAR) systems that have been developed on various mobile devices. Many of the MVS systems that we develop in this thesis are suitable for near real-time MAR applications, and the advantages of our MAR systems compared to other MAR systems are clearly identified.

2.1 Local Feature Extraction

When matching two images containing the same objects, the common objects usually appear at different positions, orientations, and scales in the two images. The objects can also be partially occluded and surrounded by background or foreground clutter. For example, Figure 2.1(a-b) show two images of the same church facade and two images of the same book cover. To robustly match a pair of images like these, local



(a) (b)

(c) (d)

(e) (f)

Figure 2.1: (a-b) Pairs of images showing the same objects. (c-d) Local feature keypoints overlaid on top of images. (e-f) Matching feature correspondences between the images connected by lines.

scale- and rotation-invariant image features can be extracted independently from each image, as shown in Figure 2.1(c-d) for scale-invariant feature transform (SIFT) features [133, 134]. Then, the corresponding features are matched between the two images, as shown in Figure 2.1(e-f). The extraction of local image features is typically carried out in two major stages: (i) feature keypoint detection, followed by (ii) feature descriptor computation. The next two sections describe the most popularly used methods for keypoint detection and descriptor computation.

Keypoint detector	Scale-invariant
Moravec detector [154]	No
Harris detector [89]	No
Shi-Tomasi detector [191]	No
Smallest univalue segment assimilating nucleus (SUSAN) [196]	No
Scale-normalized Laplacian and Hessian [127]	Yes
Scale-invariant feature transform (SIFT) [133, 134]	Yes
Maximally stable extremal regions (MSER) [141, 159]	Yes
Harris-affine detector [143, 144]	Yes
Geometry-based affine-invariant detector [212]	Yes
Intensity-based affine-invariant detector [212]	Yes
Kadir-Brady detector [111]	Yes
Speeded up robust features (SURF) [20, 18]	Yes
Features from accelerated segmented test (FAST) [179, 180]	No
Binary robust invariant scalable keypoint (BRISK) [119]	Yes
Nonlinear diffusion filtering (KAZE) [4]	Yes
Temporally coherent keypoint detector (TCKD) [136, 137]	Yes
Rotation-invariant fast features (RIFF) [204]	Yes

Table 2.1: Important sparse keypoint detectors listed in chronological order of their first appearance in the literature.

2.1.1 Feature Keypoint Detection

Feature keypoints in an image can either be chosen sparsely from salient regions [154, 89, 191, 196, 127, 133, 141, 144, 212, 134, 111, 20, 18, 159, 180, 119, 204] or densely from a regular sampling grid [120, 161, 211, 205, 219, 136, 137]. Hybrid methods that combine the advantages of sparse and dense sampling schemes have also been proposed [210, 213]. Generally, sparsely detected keypoints are more efficient to compute and thus more widely used in low-latency visual search applications, while densely sampled keypoints have found applications in state-of-the-art object classification algorithms. Since our work requires very fast feature extraction, the remainder of this section focuses on methods for sparse keypoint detection. Important sparse keypoint detectors are listed in Table 2.1 in the chronological order of their first appearance in the literature.

One of the first keypoint detectors was the corner detector developed by Moravec for robotic obstacle avoidance and navigation [154]. Subsequently, Harris and Stephens created their well-known corner detector for feature tracking [89], which finds corner regions that have large eigenvalues in the local structure matrix. These types of corners are closely related to the "good features to track" proposed by Shi and Tomasi [191]. Smith and Brady designed the smallest univalue segment assimilating nucleus (SUSAN) detector, which can locate distinctive keypoints without computing image derivatives [196].

The aforementioned keypoint detectors all lack scale-invariance. Lindeberg presented an algorithm for automatic scale selection based on maximization of the scale-normalized Laplacian or the scale-normalized Hessian [127]. An affine-invariant extension to the Harris detector has been proposed by Mikolajczyk and Schmid [143, 144]. Matas et al. searched for maximally stable extremal regions (MSERs) in an image from connected components whose areas change the least when an intensity threshold is varied [141]. Tuytelaars and Van Gool proposed two affine-invariant region detectors, one based on edges and the other based on pixel intensities [212]. Kadir et al. found affine-invariant regions by searching for image regions with high saliency values, where saliency is defined as the product of the local intensity distribution's entropy and the distribution's rate of change over different scales [111]. A performance comparison of the repeatability of several affine-invariant detectors under common image distortions is provided by Mikolajczyk et al. [146]. To mitigate the errors in scale estimation, Wu et al. [218] and Hassner et al. [90] have proposed to extract multiple keypoints at different scales centered at the same location. Instead of using linear Gaussian filtering to construct an image scale space, Alcantarilla et al. [4] perform nonlinear diffusion filtering to construct a scale space where the blurring is locally adaptive to the image data, at the expense of introducing more complex filtering operations.

In addition to improving the keypoint detector's repeatability in the presence of image distortions, improving the keypoint detector's efficiency has been a major focus of many recent works. In his well-known scale-invariant feature transform (SIFT) algorithm [133, 134], Lowe developed an efficient method of detecting extrema in a

difference-of-Gaussian (DoG) scale-space, as an approximation to detecting extrema in a Laplacian-of-Gaussian (LoG) scale-space. Seeking an even faster detector, Bay et al. used box filters in the speeded up robust features (SURF) algorithm to approximate the scale-normalized Hessian response [20, 18]. The advantage of using box filters is that the filter response can be computed using a small fixed number of operations via an integral image, irrespective of the size of the filter kernel. Nister and Stewenius developed a fast implementation of MSERs whose algorithmic complexity is linear in the number of pixels [159]. To reduce the temporal jitter of keypoints in video frames, Makar et al. created a temporally coherent keypoint detector [136, 137] by extracting SIFT keypoints in keyframes and efficiently tracking these keypoints until the next keyframe appears.

To date, one of the fastest keypoint detectors available is the features from accelerated segmented test (FAST) corner detector [179, 180]. Due to its speed, FAST corners have been utilized for feature tracking at video frame rates on a mobile device [203]. One drawback, however, is that the FAST detector is not scale-invariant. To address this deficiency, Takacs et al. developed the rotation-invariant fast features (RIFF) keypoint detector, which is fast enough to run at video frame rates on a mobile device and provides an approximation of the LoG scale-space using differences between box filter responses [204]. Leutenegger et al. developed the binary robust invariant scalable keypoint (BRISK) detector [119], which generates an image scale-space, finds FAST corners at each scale, and applies non-maximum suppression across neighboring scales.

2.1.2 Feature Descriptor Computation

Once a feature keypoint is detected in an image, a descriptor can be generated to characterize the local image patch centered at the keypoint, as depicted in Figure 2.2. The descriptor should be robust against image distortions, while at the same time being capable of distinguishing different patches from one another. Table 2.2 lists some well-known feature descriptors in the chronological order of their first appearance in the literature. Amongst these descriptors are the highly popular SIFT [133, 134] and

Feature descriptor	Uses gradients
Scale-invariant feature transform (SIFT) [133, 134]	Yes
Gradient location and orientation histogram (GLOH) [145]	Yes
Speeded up robust features (SURF) [20, 18]	Yes
Dual-tree complex wavelet transform (DTCWT) [114]	No
Compressed histogram of gradients (CHoG) [33, 34]	Yes
Machine-optimized DAISY [216, 26]	Yes
Center-symmetric local binary pattern (CS-LBP) [92]	No
Rotation-invariant fast features (RIFF) [203, 204]	Yes
Binary robust independent elementary features (BRIEF) [28, 27]	No
Compact and real-time descriptors (CARD) [7]	Yes
Oriented FAST and rotated BRIEF (ORB) [181]	No
Fast retinal keypoint descriptor (FREAK) [3]	No
Multi-scale k-jet [118]	Yes
Local quantized pattern (LQP) [95]	No

Table 2.2: Important image feature descriptors listed in chronological order of their first appearance in the literature.

SURF [20, 18] descriptors. The SIFT and SURF descriptors are rotation-invariant because the orientation of the patch is chosen as the direction of a peak in a local orientation histogram. These descriptors are robust against brightness and contrast changes because intensity gradients are used for the descriptor calculation and the descriptor is normalized to have unit L_2 norm. For increased descriptiveness, each patch is split into a square grid of 4×4 spatial cells. Within each spatial cell, weighted histograms or moments of the intensity gradients contained in that spatial cell are computed to form part of the overall descriptor. For resilience against localization errors, soft binning of intensity gradients between neighboring spatial cells is applied.

A detailed performance evaluation of local descriptors by Mikolajczyk and Schmid showed that gradient-based descriptors like SIFT or SURF obtain the best performance [145]. Other gradient-based descriptors include the gradient location and orientation histogram (GLOH) from Mikolajczyk and Schmid [145], which uses a polar grid for the spatial cells as opposed to the square grid used in SIFT and SURF. To form a descriptor which is both discriminative and compact, Chandrasekhar et

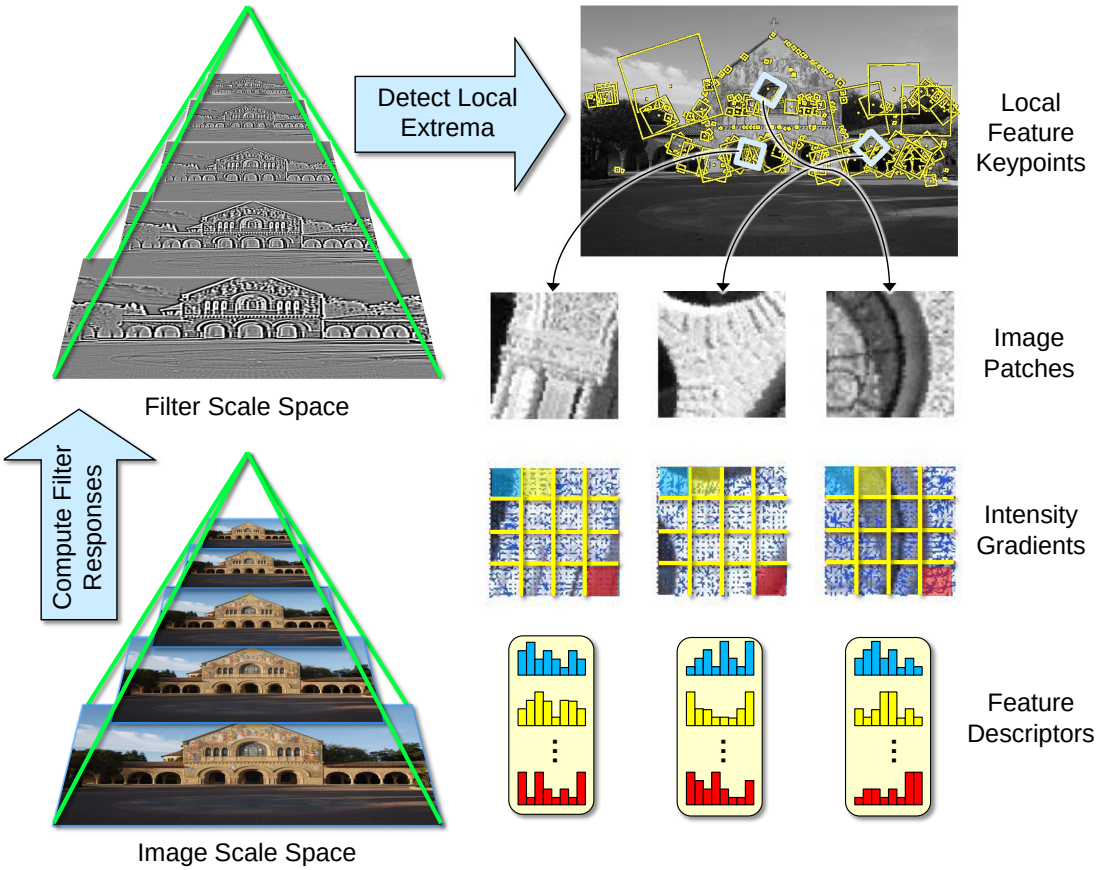


Figure 2.2: Detection of feature keypoints from local extrema in a scale space, and computation of feature descriptors from intensity gradients within spatial cells of local image patches.

al. developed the compressed histogram of gradients (CHoG) descriptor [33, 34] by efficiently compressing the gradient distribution in each spatial cell and comparing the distributions using the Kullback-Leibler divergence or the Monge-Kantorovich distance [112]. In particular, the Monge-Kantovorich distance, also known as the transportation distance, considers the least amount of work needed to transform one distribution into another distribution. The RIFF descriptor created by Takacs et al. [203, 204] also uses a histogram of gradients, but computes radial and tangential gradients rather than the conventional horizontal and vertical gradients. Ambai and Yoshida presented the compact and real-time descriptors (CARD) [7], which first uses

lookup tables to compute histograms of gradients over arbitrary spatial cell configurations and then uses sparse hashing to create a compact binary descriptor. Larsen et al. [118] created a descriptor based on the local multi-scale k-jet [72], which contains higher-order spatial derivatives of a Gaussian-filtered image.

Alternative descriptors which do not use intensity gradients have also been proposed. Kingsbury used the coefficients of the dual-tree complex wavelet transform (DTCWT) as a scale-and-rotation-invariant descriptor [114]. Heikkila et al. proposed the center-symmetric local binary pattern (CS-LBP) descriptor [92] as an extension of the local binary pattern (LBP) texture operator [162]. In a similar spirit, Hussain and Triggs developed the local quantized pattern (LQP) descriptor [95]. Calonder et al. created a binary descriptor called binary robust independent elementary features (BRIEF) by comparing the intensities at different pairs of locations in the patch and encoding the comparison results into a binary vector [28, 27]. Rublee et al. combined an oriented version of the FAST detector with the BRIEF descriptor to create the oriented FAST and rotated BRIEF (ORB) descriptor [181]. Alahi et al. used a human-vision-inspired retinal sampling pattern and pixel intensity comparisons to create the fast retinal keypoint (FREAK) descriptor [3].

Recently, Heinly et al. conducted a performance evaluation of the newer binary feature descriptors and reported that (i) the binary descriptors have slightly lower pairwise matching accuracy than gradient-based descriptors, and (ii) the binary descriptors are noticeably faster to extract and match than the gradient-based descriptors. [93]. Takacs et al. found that binary descriptors have much lower image retrieval accuracy for large databases than gradient-based descriptors [204]. Due to the importance of achieving high accuracy in large-scale retrieval for building MVS systems, the research and experiments presented in this thesis will use gradient-based descriptors.

2.2 Pairwise Image Matching

The local feature sets extracted from two images can be compared to determine if these two images share some common object(s). The pairwise matching process should be robust against extraneous, missing, and noisy features in the two sets and

provide correspondences between the matching features. Typically, pairwise matching is carried out in two stages: (i) nearest neighbor descriptor matching and (ii) geometric verification of feature correspondences. The next two sections discuss popular methods in the literature for these two topics.

2.2.1 Nearest Neighbor Descriptor Matching

The problem for nearest neighbor (NN) descriptor matching is the following. Denote the sets of descriptors for the first and second images by $\mathbf{V}_1 = \{\mathbf{v}_{1,1}, \cdots, \mathbf{v}_{1,N_1}\}$ and $\mathbf{V}_2 = \{\mathbf{v}_{2,1}, \cdots, \mathbf{v}_{2,N_2}\}$, respectively. For each descriptor $\mathbf{v}_1 \in \mathbf{V}_1$, we want to find the nearest descriptor (NN) $\mathbf{v}_2^{NN} = \text{argmin}_{\mathbf{v}_2 \in \mathbf{V}_2} d(\mathbf{v}_1, \mathbf{v}_2)$ with respect to some distance $d(\mathbf{v}_1, \mathbf{v}_2)$. Typically, the Euclidean distance is used to compare descriptors, although most of the methods we describe in this section apply to any positive power of a metric. Assuming the descriptor has dimensionality l, the complexity of a brute-force search for NNs is $O(l\, N_1\, N_2)$, which could lead to undesirably long computational delays when the NN search is performed between the query image and every database image in a large database.

Since visual features are inherently noisy, finding the NN in the second image for a feature descriptor from the first image does not ensure finding a correctly matching feature correspondence between the two images. Hence, expending a great amount of effort to find the NN for each feature descriptor may not lead to higher image matching performance. To reach an effective compromise between the computational speed and the feature matching accuracy, finding an approximate nearest neighbor (ANN) has become a popular alternative to finding the exact NN. Arya and Mount [12] and Arya et al. [13] have shown that if the requirement for finding the NN $\mathbf{v}_2^{NN} = \text{argmin}_{\mathbf{v}_2 \in \mathbf{V}_2} d(\mathbf{v}_1, \mathbf{v}_2)$ is relaxed to finding an ANN $\mathbf{v}_2^{ANN}$ such that $d\left(\mathbf{v}_1, \mathbf{v}_2^{ANN}\right) \leq (1+\epsilon)\, d\left(\mathbf{v}_1, \mathbf{v}_2^{NN}\right)$, then the complexity can be reduced to $O(c_{l,\epsilon}\, N_1 \log N_2)$ where $c_{l,\epsilon} \leq l\,(1 + 6l/\epsilon)^l$. Their ANN algorithms use a K-D tree [77, 21, 197] to partition the vector space. Efficient search of ANNs in sub-linear time has been extensively studied with other tree structures, including B trees [60], K-D-B trees [178], R trees [88], SR trees [113], and M trees [58].

For image matching applications, ANN methods are attractive because they can establish a set of tentative feature correspondences between two images without incurring the long computational delays of exact NN methods. Thus, multiple alternative methods for ANN have appeared in the computer vision literature. Lowe developed a best bin first (BFF) algorithm for ANN search of SIFT descriptors [134]. Muja and Lowe subsequently created the fast library for ANN (FLANN) algorithm by using multiple randomized K-D trees [155]. Olonetsky and Avidan created the k-d tree coherent ANN (TreeCANN) algorithm specifically for sparse sets of patches in images [163]. Jegou et al. utilized product quantizers and look-up tables for fast ANN searching of feature descriptors [105].

To propose tentative feature correspondences between two images, Lowe proposed the following distance ratio test [134]. For a descriptor $\mathbf{v}_1 \in \mathbf{V}_1$, if $\mathbf{v}_2^{NN,1}$ and $\mathbf{v}_2^{NN,2}$ are the first and second nearest descriptors in $\mathbf{V}_2$, then the correspondence $\left(\mathbf{v}_1, \mathbf{v}_2^{NN,1}\right)$ is retained for further processing if and only if $d\left(\mathbf{v}_1, \mathbf{v}_2^{NN,1}\right) \leq r \cdot d\left(\mathbf{v}_1, \mathbf{v}_2^{NN,2}\right)$ for some ratio $0 < r < 1$. Typically, $r \in [0.7, 0.9]$ yields good matching performance for many different types of descriptors.

2.2.2 Geometric Verification

After nearest neighbor descriptor matching, some of the feature correspondences between the two images are incorrectly established. These outlier feature correspondences will be inconsistent with respect to a valid geometric model that describes the relationship amongst the inlier feature correspondences. The role of geometric verification is to remove the outlier feature correspondences, while at the same time estimating the parameters of the underlying geometric model for the inlier feature correspondences between the two images. Commonly used geometric models include full 3D object models [17], epipolar models [57], and projective and affine models [171, 207].

The random sample consensus (RANSAC) algorithm [71] is often used to estimate the geometric model robustly when the number of outlier feature correspondences is a significant fraction of the total number of feature correspondences. Variants

of RANSAC adapted to improve the matching in computer vision applications include maximum likelihood estimation sample consensus (MLESAC) [206], preemptive RANSAC (P-RANSAC) [157], randomized RANSAC (R-RANSAC) [48, 51], progressive sample consensus (PROSAC) [49], RANSAC for quasi-degenerate data (QDEGSAC) [74], RANSAC with groupings (GroupSAC) [156], stable RANSAC (StaRSAC) [45], and spatially consistent RANSAC (SCRAMSAC) [186].

2.3 Large-Scale Image Retrieval

When querying a large database of images, the pairwise matching methods described in the last two sections are too computationally expensive to apply between the query image and every database image. The main task in large-scale image retrieval is to develop a method of quickly searching a large database to produce a ranked shortlist of database candidates, which includes the correct database images. Typically, the database is preprocessed and organized into an indexing structure for fast querying later on. In the past decade, the most popular and successful indexing structures have utilized *feature histograms*. More recently, some compact indexing structures have instead employed *feature residuals*. In the next two sections, we review the relevant literature on histogram-based and residual-based image retrieval. Table 2.3 lists the important contributions to image retrieval for both types of methods.

2.3.1 Feature Histograms

Sivic and Zisserman first used feature histograms in retrieving frames that showed a particular object from full-length movies [193]. For feature description, they used two different types of keypoint detectors – Harris-affine [143, 144] and MSER [141] – and the SIFT descriptor [134] with both types of keypoints. For indexing, they constructed moderately sized codebooks containing 6K and 10K codewords for the Harris-affine SIFT and MSER SIFT features, respectively, by k-means clustering on a set of training feature descriptors. The codewords in each codebook are commonly called *visual words* in the computer vision literature. For each query image,

Retrieval method	Histogram	Residual
Flat feature histograms [193]	×	
Pyramid match kernel (PMK) [83, 84, 86, 85]	×	
Vocabulary tree and inverted index [158]	×	
Spatial contextual weighting [140, 223, 214]	×	
Greedy multi-path search [187]	×	
Contextual dissimilarity measure [107, 109]	×	
Multiple randomized k-d trees [171]	×	
Fisher kernels for GMMs [167]		×
Query expansion [55, 53, 10]	×	
Adaptive vocabulary forests [220]	×	
Histogram soft binning [172]	×	
Min hash [56, 54, 52]	×	
Hamming embedding [100, 97]	×	
Vocabulary tree reranking [100, 104, 208]	×	
Feature burstiness discounting [102]	×	
Bundled features [217, 130]	×	
Multi-indexing [218, 215, 15]	×	
Mini bag-of-features [103]	×	
Compressed Fisher kernel (CFK) [168, 169, 185]		×
Vector of locally aggregated descriptors (VLAD) [106, 108]		×
Spatial histograms [29, 126, 227]	×	
Very fine visual vocabulary [147]	×	
Fisher kernels for non-i.i.d. models [59]		×
Adapation, normalization, spatial VLAD [11]		×
Histogram PCA and whitening [99]	×	
Generalized IDF weights [226]	×	

Table 2.3: Important large-scale image retrieval methods listed in chronological order of their first appearance in the literature.

the database images are scored using the popular term-frequency inverse-document-frequency (TF-IDF) framework borrowed from the information retrieval literature [182, 184]. After a shortlist of the top few hundred database candidates is determined by TF-IDF scoring, more precise nearest neighbor descriptor matching and geometric verification can be applied to just the images in the shortlist.

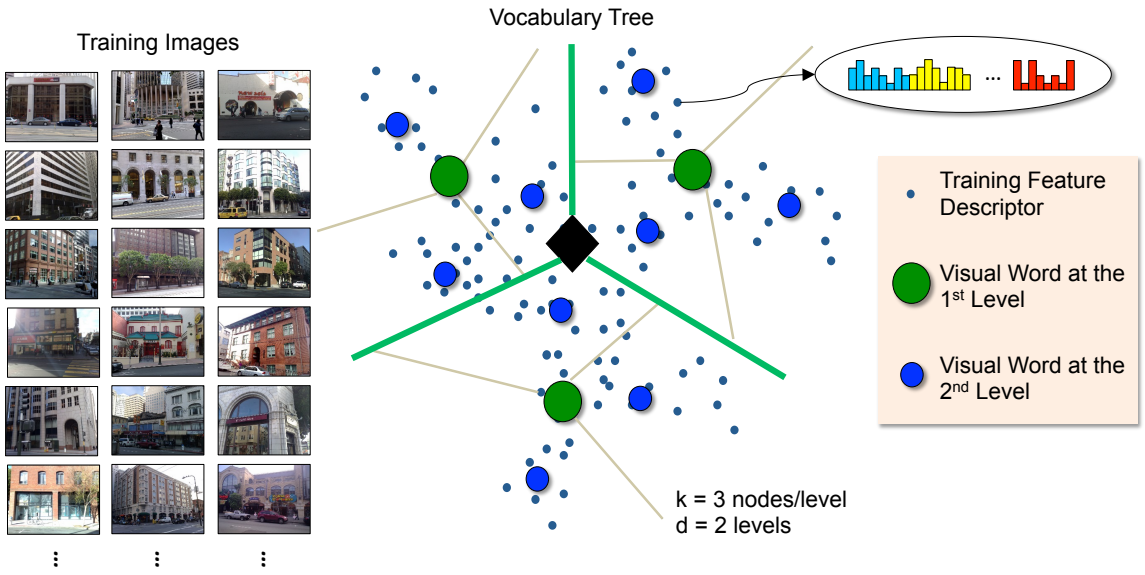


Figure 2.3: Hierarchical k-means clustering for a set of training feature descriptors.

Since retrieval performance generally improves when the codebook size increases, Nister and Stewenius proposed using much larger codebooks, containing up to 1M codewords [158]. Their work is similar to that of Grauman and Darrell [83, 84, 85, 86], although Grauman and Darrell focused more on image classification rather than image retrieval. Construction of large codebooks is possible with hierarchical k-means clustering. The output of the hierarchical k-means clustering procedure is a tree-structured vector quantizer (TSVQ) [25, 46, 177, 61, 87, 122] that is commonly called a *vocabulary tree* in the computer vision literature [158]. Figure 2.3 shows a toy example of hierarchical k-means clustering with a branch factor of $k = 3$ nodes/level and a depth of $d = 2$ levels. Figure 2.4 shows the corresponding vocabulary tree. With a vocabulary tree, we can create a global image signature in the form of a tree histogram, which records how often each leaf node in the vocabulary tree is visited by an image's feature descriptors. Figure 2.4 shows the tree histograms for a query image and several database images, including a database image that correctly matches the query image, for the same toy example with $k = 3$ and $d = 2$. In practice, the parameters $k = 10$ and $d = 6$ have been shown to provide good retrieval performance [158], which correspond to a vocabulary tree with 1M leaf nodes.

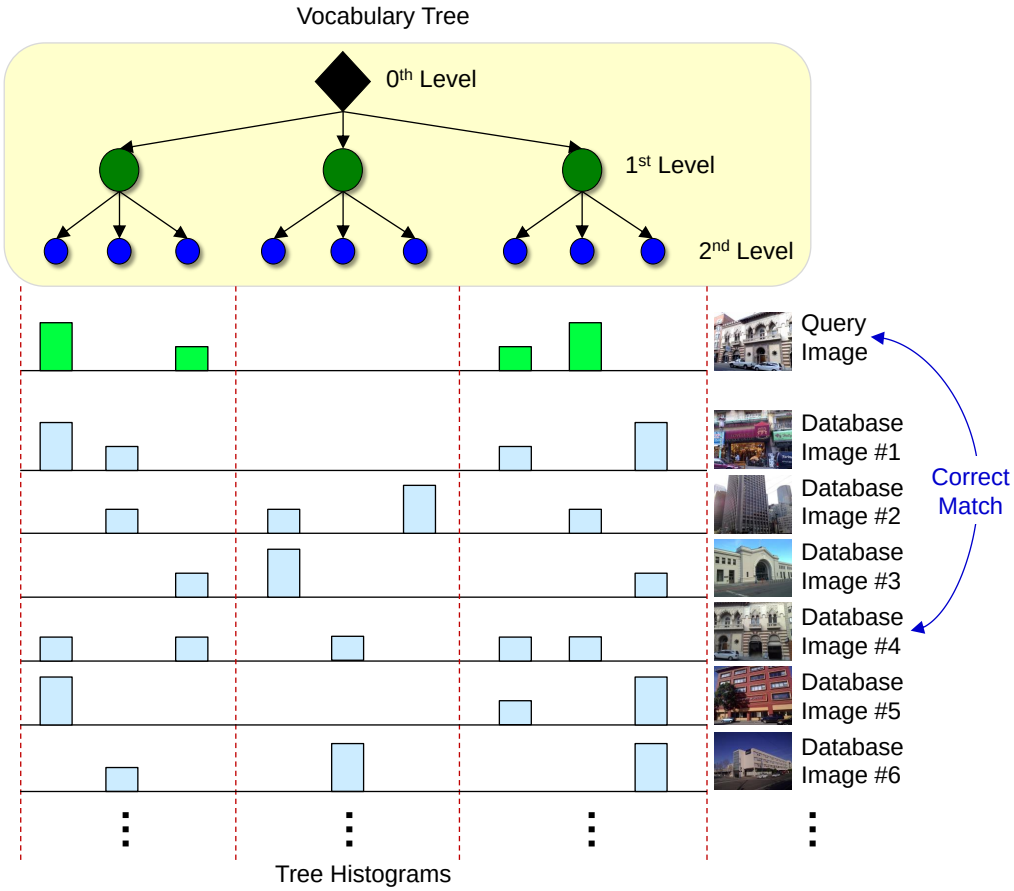


Figure 2.4: Vocabulary tree and associated tree histograms for a query image and several database images.

Nister and Stewenius also utilized an *inverted index* to speed up the score computations [158]. The inverted index is another useful tool borrowed from information retrieval [153, 229]. Figure 2.5 shows the inverted index associated with the vocabulary tree from Figure 2.4. For each visual word in the codebook, the inverted index stores a list of the identifiers (IDs) of all database images which have at least one feature quantized to that visual word. Additional information stored in the inverted lists depends on the specific retrieval system and may include counts of how often each image visited the leaf node [158], IDs of which features in each image visited [208], hashes of the visiting feature descriptors [100], and associated geometric quantities [100, 207]. For a large codebook, e.g., 1M visual words, and several hundred

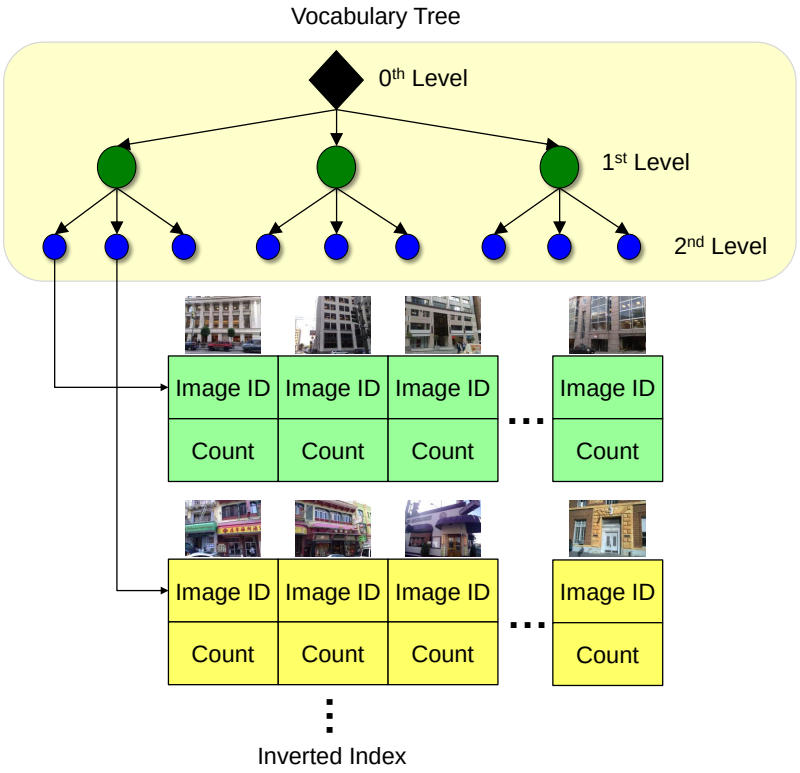


Figure 2.5: Inverted index associated with a vocabulary tree. Each leaf node of the vocabulary tree is connected to an inverted list of image identifiers and feature counts.

local features per image, the number of entries in each inverted list is much smaller than the total number of database images. Thus, traversing the inverted lists reached by the several hundred features of a query image is computationally efficient, and incremental updates to the TF-IDF scores of all database images can be performed while selectively traversing a small number of inverted lists.

Many effective improvements to the basic feature histograms framework have been proposed in the literature. Schindler et al. searched multiple paths through the vocabulary tree greedily before deciding the closest leaf node for a feature descriptor [187]. To mitigate the effects of quantizing matching features to different cells, Philbin et al. quantized a feature descriptor to several nearest leaf nodes with a soft binning scheme [172], as opposed to the hard binning scheme used by Nister and Stewenius [158]. Another method for reducing errors caused by quantization is to use a Hamming

embedding within each Voronoi cell to determine if two descriptors quantized to the same Voronoi cell represent matching or non-matching features [100]. Recently, an asymmetric version of Hamming embedding has been presented [97]. Philbin et al. used multiple randomized k-d trees to partition the descriptor space [171] instead of using a single vocabulary tree as in [158]. Jegou et al. defined a contextual dissimilarity measure to account for the nonuniform spacing of tree histograms [107, 109]. In subsequent work, Jegou et al. effectively discounted the influence of bursty features in tree histogram voting [102] and performed principal component analysis (PCA) and whitening on tree histograms [99]. Zheng et al. generalized the IDF weights to improve retrieval [226]. Mikulik et al. proposed methods for learning a very fine vocabulary [147]. Yeh et al. adapted the vocabulary over time in response to changing feature statistics [220]. Simultaneous indexing of multiple complementary feature descriptors has also been studied by Wu et al. [218], Wengert et al. [215], and Babenko and Lempitsky [15].

So far, the discussion has focused on histogram-based methods that use only the feature descriptors. Incorporation of geometric information for histogram-based retrieval has also received significant attention. Co-occurrence of features in local configurations has been studied by Sivic and Zisserman [194], Quack et al. [175], Chum and Matas [50], Jamieson et al. [98], Yuan et al. [221], and Liu and Yan [129]. Li et al. combined a visual codebook with local spatial proximity distributions [128]. Wu et al. [217] and Liu et al. [130] bundled multiple SIFT features that lie inside an MSER and exploited the consistency of features within each bundle for database re-ranking. Cao et al. [29], Lin and Brandt [126], and Zhou et al. [227] created spatial extensions to the feature histogram by exploiting weak geometric consistency relationships. Weak similarities in scale, orientation, or distance ratios among visual word matches are also utilized by Jegou et al. [100, 104] and Tsai et al. [208] to re-rank database candidates. Marszaek and Schmid [140], Zhang et al. [223], and Wang et al. [214] applied different forms of contextual weighting based on local spatial relationships. Chum et al. [55, 53] and Arandjelovic and Zisserman [10] applied query expansion to features in matching spatial regions of an image. Philbin et al. developed a geometric latent dirichlet allocation model for groups of features [173].

Zhang et al. created geometry-preserving visual phrases as an extension to single visual words [225].

As the size of the database grows, feature histograms consume a significant amount of memory. On a server where many tasks are running concurrently or a mobile device with a small random access memory (RAM) capacity, the memory usage of feature histograms can become a performance bottleneck. Index compression techniques have been extensively studied in information retrieval [23, 170, 24, 151, 152, 22, 188, 78, 192, 8, 149, 9, 150, 222], but these techniques primarily focus on lossless compression of quantities derived from text documents. For memory-efficient image retrieval, lossy compression methods that consider image feature statistics and optimize for the final image retrieval objectives can provide much greater memory savings. Jegou et al. created mini-BOFs as memory-efficient replacements for full tree histograms, although the mini-BOFs yield lower accuracy than tree histograms [103]. Zhang et al. created a compact index by discarding non-discriminative features and indexing the remaining features with locality sensitive hashing (LSH) [224]. Chum et al. have developed compact min hashes for visual codebooks [56, 54, 52], although these methods have lower recall than feature histograms.

2.3.2 Feature Residuals

Due to the high memory usage of feature histograms, a new trend in computer vision is the development of compact and discriminative feature residuals. The residual representation employs a small codebook of visual words, typically containing 128-256 codewords. A histogram over such a small codebook would not be sufficiently discriminative for large-scale image retrieval. Instead, the primary signals used to generate discriminative signatures are now the quantization errors or residuals that exist between the feature descriptors and their nearest codewords. Figure 2.6 shows a toy example with a codebook of three visual words, the feature descriptors for three images quantized to the nearest visual words, and the formation of the corresponding feature residuals. Within each Voronoi cell, the residuals for each image

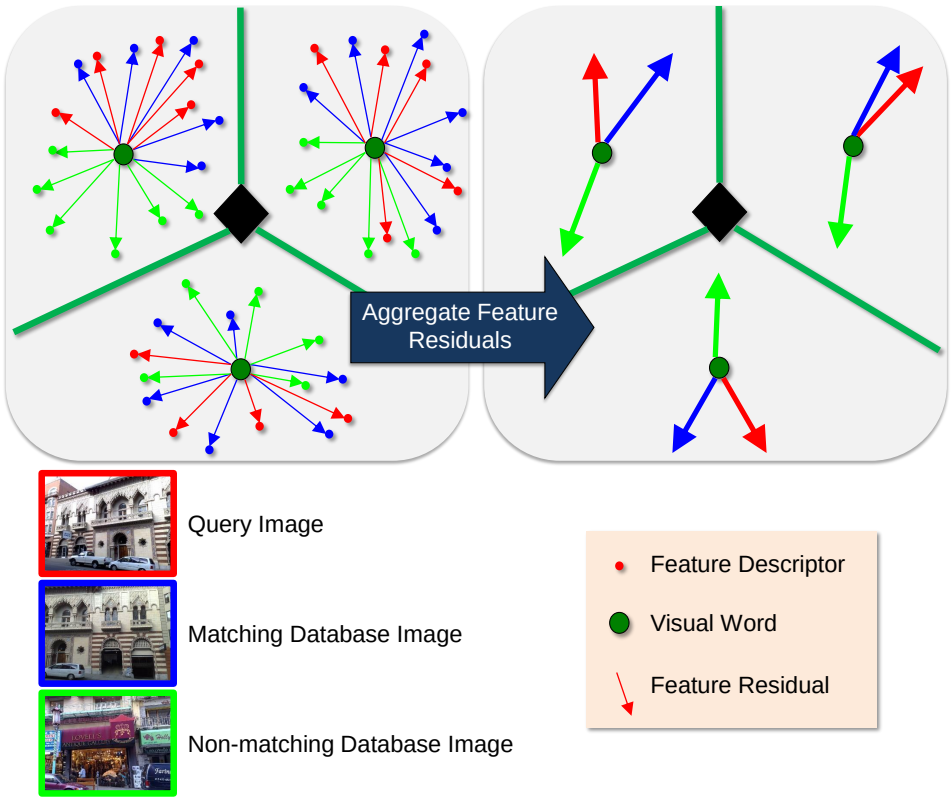


Figure 2.6: Generation and aggregation of feature residuals for three images, using a small codebook composed of three visual words.

are aggregated, e.g., into a mean residual vector. It can be observed that the aggregated residuals of the query image are more similar to the aggregated residuals of the matching database image than to the aggregated residuals of the non-matching database image.

Quantization residuals have been used in discriminant vector quantization (DVQ) [121, 164, 165]. Given a set of training data, DVQ trains a two-stage vector quantizer. In the first stage, each vector is quantized to 1 of M classes. Then, in the second stage, the residual between the vector and its assigned class centroid is quantized.

A soft-binned version of feature residuals leads to a Fisher kernel formulation. Initially, Fisher kernels for Gaussian mixture models (GMMs) were developed by Perronnin and Dance for image classification [167]. Perronnin et al. subsequently created an improved version of Fisher kernels for image retrieval [168] and image

classification [169]. Around the same time, Jegou et al. developed a hard-binned version called the vector of locally aggregated descriptors (VLAD), which achieved impressive reductions in memory usage. Douze et al. improved the VLAD representation as part of a system for video retrieval [64]. Later, Douze et al. combined Fisher kernels with image attributes learned by support vector machines (SVMs) for more accurate image retrieval [65]. Sanchez and Perronnin developed hashed-based and quantization-based methods for compressing high-dimensional Fisher kernels [185]. A comparison of different versions of Fisher kernels and VLAD is presented in [108]. More recently, an non i.i.d. image model for the Fisher kernel has been proposed by Cinbis et al. [59]. Arandjelovic and Zisserman presented three improvements to VLAD: vocabulary adaptation, per-cell L_2 normalization, and a spatial VLAD representation for object localization [11]. Jegou et al. showed that using multiple codebooks, PCA, and whitening can further improve performance [99].

In the subsequent chapters, we will study how to generate both compressed feature histograms and feature residuals for representing large databases of images, and we will perform a detailed performance evaluation of compressed feature histograms and feature residuals. This research will then inform us how to best use feature histograms or feature residuals in building practical MVS systems.

2.4 Mobile Augmented Reality Systems

Mobile augmented reality (MAR) systems augment a user's view of the physical world with additional information about recognized objects. The earliest MAR systems were implemented with head-mounted displays. Sutherland developed a MAR system [199, 200] that used a head-mounted display to show an augmented view of a room to the user. A head-position sensor connected to the ceiling was used to measure the position and orientation of the user's head, although the user could still freely walk around the room. Smailagic and Siewiorek created a MAR system called Navigator, which provides navigation assistance throughout the Carnegie Mellon University campus [195]. Their system attached a computer to a person's back, displayed text and graphics about nearby buildings on a head-mounted display, and parsed voice

commands by speech recognition. Mann also created a head-mounted display capable of augmenting the screen with textual information relevant to recognized objects in the user's field-of-view [138]. The computer in Mann's system was also worn in the back. Feiner et al. created a "touring machine" consisting of a 3D head-mounted display, a backpack unit for power and computation, and a handheld 2D display [70]. Labels for nearby buildings are augmented in the 3D display, while a web page about each recognized building is shown on the 2D display.

A second generation of head-mounted MAR displays has featured devices that are easier to wear like normal glasses. Google Glass provides a monocular display for the right eye, a camera, and controls through a tap interface and voice commands [81]. MAR applications currently supported on Google Glass include an outdoor exercise tracker, a cooking assistant, and a foreign word translator. The Meta Metapro includes a binocular display for 3-d holographic vision and a depth camera [142]. Many MAR applications have been developed already for Metapro, including a personal document annotator, a 3-d virtual sculptor, and an interior design assistant. Similarly, the Epson Moverio provides a 3-d binocular see-through display that is controlled by a small handheld controller [67]. This device is intended primarily as an augmented gaming device. All of these new head-mounted displays are connected to WiFi networks, enabling them to easily download virtual content and information from online databases for augmentations.

At the same time, the quickly growing availability of smartphones and tablets has fueled the increasing popularity of many MAR applications on these mobile devices. Examples include Google Goggles [82], Nokia Point-and-Find [160], Amazon Flow [6], and Kooaba AR [115]. These applications can recognize objects such as product packages, barcodes, billboards, artwork, text documents, store signs, and building facades. Other MAR applications have been developed for text translation [96], road sign translation [189], plant leaf identification [116], museum guidance [73, 19, 148, 110], site tour guidance [202, 62, 68], data center assistance [63], children's book animation [166], interactive gaming [94, 1], and computational rephotography [2].

To recognize objects in the aforementioned systems, a query image captured through the mobile device's camera is typically compared against a database in the

cloud, and the image comparisons utilize the local image features described in Sections 2.1 and the image matching methods described in Section 2.2- 2.3. When the photo is taken outdoors, the mobile device's GPS coordinates can be used to constrain the database search to a local neighborhood [202]. The mobile device's gyroscope can also be used to estimate the direction of gravity, so that the query photo can be rotated to an upright orientation and more discriminative upright local features can be extracted [117, 14].

Our work in this thesis includes extensive experiments in the design and development of robust, low-latency MAR applications on mobile devices. Whereas existing MAR systems need to contact a remote server to perform the image comparisons and are hence prone to fluctuations in network quality or server load, we instead show that storing an image database directly on the mobile device can lead to persistently fast and reliable recognition. Based on the memory-efficient databases that we develop in Chapters 3 - 4, we will create a MAR system in Chapter 5 that can accurately and quickly recognize a variety of objects – including media covers, book spines, artwork, buildings, and video frames – regardless of external network or server conditions. Recognition results appear almost instantly in the mobile device's viewfinder, and the augmentation layers closely track the actual movements of the recognized objects in the viewfinder for as long as the user focuses on these objects. Our MAR system is additionally capable of automatically inferring the user's interest in different parts of the scene, so the user never has to press a button to initiate a query, and of incrementally updating the image database contents through low-bitrate query expansions into the cloud.

Chapter 3

Memory-Efficient Feature Histograms

The vast majority of image retrieval methods today rely on feature histograms, as shown in Section 2.3. Feature histograms can be generated and compared efficiently, and they offer high retrieval accuracy even for large image databases. The primary drawback of feature histograms is that they consume a significant amount of memory, which can easily lead to a performance bottleneck when the retrieval system is running on a server with many concurrent memory-intensive tasks or running on a mobile device with a small memory capacity.

In this chapter, we develop two methods that substantially reduce the memory usage of feature histograms, with no adverse effect on the retrieval accuracy.[1] Section 3.1 first describes how to generate tree histograms formed over the nodes of a large vocabulary tree, and then describes how to efficiently compare these histograms by a weighted histogram intersection method. To reduce the memory usage of the database, Section 3.2 explains how to effectively encode the database tree histograms by a method called tree histogram coding (THC). For fast querying, the database tree histograms are often stored implicitly in an inverted index. Therefore, Section 3.3 focuses on another method called inverted index coding (IIC) to substantially compress the inverted index associated with a vocabulary tree. Both THC and IIC can

[1] Preliminary results of our work have appeared in [38, 39].

Symbol	Meaning
d	Depth of a vocabulary tree
k	Branch factor of a vocabulary tree
l	Dimensionality of a feature descriptor
r	Number of most promising tree paths searched per descriptor
m	Number of soft-binned leaf node assignments per descriptor
N_{feat}	Number of local features per image
$\mathbf{n}_{i,j}$	j^{th} node at i^{th} level in vocabulary tree
$C(\mathbf{n}_{i,j})$	Children at $(i+1)^{\text{st}}$ level of j^{th} node at i^{th} level in vocabulary tree
$d_{i,j}$	Euclidean distance between a descriptor and node $\mathbf{n}_{i,j}$
$w_{i,j}$	Soft-binned count at node $\mathbf{n}_{i,j}$ for a descriptor
$c_{i,j}$	Total feature count at node $\mathbf{n}_{i,j}$ for an image
$\mathbf{h}$	All leaf node feature counts for an image
$J(\mathbf{h})$	Identifiers for positive-count leaf nodes for an image
$R(\mathbf{h})$	Identifier runlengths of positive-count leaf nodes for an image
$C(\mathbf{h})$	Counts for positive-count leaf nodes for an image
$\widehat{C}(\mathbf{h})$	Quantized counts for positive-count leaf nodes for an image
$s(\mathbf{h}_1, \mathbf{h}_2)$	Histogram intersection between histograms $\mathbf{h}_1$ and $\mathbf{h}_2$
$w_{\text{idf}}(\mathbf{n}_{i,j})$	Inverse document frequency (IDF) weight for node $\mathbf{n}_{i,j}$
$N_{\text{idf}}(\mathbf{h})$	IDF-weighted normalization factor for an image
N_{db}	Total number of database images
$N_{i,j}$	Number of database images that have visited node $\mathbf{n}_{i,j}$
H_J	Single-symbol entropy for node or image identifiers
H_R	Single-symbol entropy for node or image runlengths
N_c	Number of centroids used by a scalar quantizer
N_{zones}	Number of zones used for feature count quantization
$q(c)$	Index of the centroid to which a count c is quantized
C_i	Set of counts quantized to the i^{th} centroid
$b(x, s)$	Binary encoding of integer x with s bits of precision
$\mathbf{i}_u$	Inverted list for leaf node $\mathbf{n}_{d-1,u}$
M_u	Length of inverted list for leaf node $\mathbf{n}_{d-1,u}$
$G(M)$	Ordering gain for a set of M symbols
p_{feat}	Probability that a leaf node is visited by a feature
p_{image}	Probability that a leaf node is visited by an image

Table 3.1: Mathematical symbols used in Chapter 3.

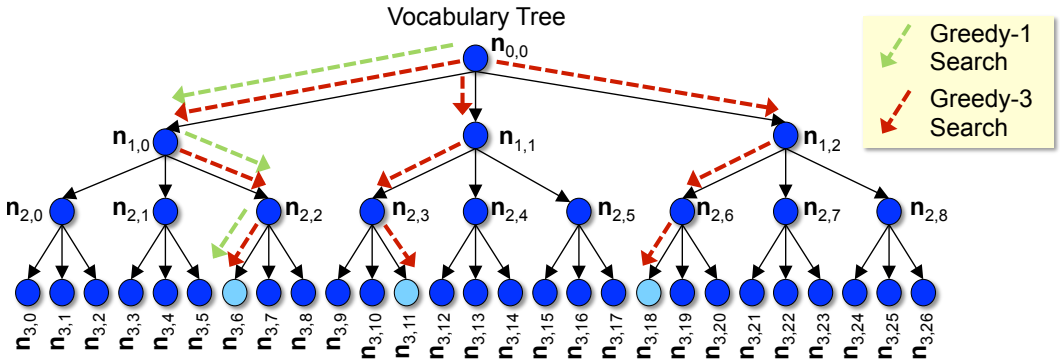


Figure 3.1: Greedy search through a vocabulary tree for the nearest leaf node, with greedy-1 and greedy-3 search options.

achieve several-fold reductions in the memory usage of a large database while maintaing the same retrieval accuracy as the original uncompressed tree histograms, as demonstrated by large-scale retrieval results in Section 3.4. A fast decoding capability is also designed into both THC and IIC, so these methods can be easily incorporated into low-latency MVS systems. For the reader's reference, all mathematical symbols used in this chapter are listed in Table 3.1.

3.1 Histogram Generation and Comparison

3.1.1 Greedy Multi-Path Search

Suppose a vocabulary tree has depth d and branch factor k. For example, Figure 3.1 shows a vocabulary tree with the parameters $d = 4$ and $k = 3$. The nodes at the i^{th} level ($i = 0, 1, \cdots, d-1$) of the tree are enumerated as $\mathbf{n}_{i,0}, \cdots, \mathbf{n}_{i,k^i-1}$, where each node $\mathbf{n}_{i,j} \in \mathbb{R}^l$ and l is the dimensionality of the chosen image feature descriptor, e.g., $l = 128$ for SIFT. Except at the leaf level $i = d-1$ where nodes have no children, the children of a node $\mathbf{n}_{i,j}$ are $C(\mathbf{n}_{i,j}) = \{\mathbf{n}_{i+1,jk}, \cdots, \mathbf{n}_{i+1,(j+1)k-1}\}$.

Quantization of feature descriptors through a vocabulary tree typically uses a greedy search procedure. The simplest case is the greedy-1 search described in Algorithm 3.1. In the algorithm, $d(\mathbf{n}, \mathbf{q})$ represents the distance between the tree node $\mathbf{n}$ and the image feature descriptor $\mathbf{q}$, e.g., the Euclidean distance. The algorithm

Algorithm 3.1 Greedy-1 search for the nearest leaf node in a vocabulary tree.

Require: Image feature descriptor $\mathbf{q} \in \mathbb{R}^l$

$\mathbf{p} \coloneqq \mathbf{n}_{0,0}$ ▷ Initialize to the root node
for level $= 1 \rightarrow d$ **do**
  $\mathbf{p} \coloneqq \text{argmin}_{\mathbf{n} \in C(\mathbf{p})} d(\mathbf{n}, \mathbf{q})$ ▷ Find the closest child of the current node
end for

Algorithm 3.2 Greedy-r search for the r nearest leaf nodes in a vocabulary tree.

Require: Image feature descriptor $\mathbf{q} \in \mathbb{R}^l$

$P \coloneqq \{\mathbf{n}_{0,0}\}$ ▷ Initialize to include the root node
for level $= 1 \rightarrow d$ **do**
  $C(P) = \cup_{\mathbf{n} \in P} C(\mathbf{n})$ ▷ Gather children of all nodes in the current search set
  $P \coloneqq \text{r-argmin}_{\mathbf{n} \in C(P)} d(\mathbf{n}, \mathbf{q})$ ▷ Find the r closest children
end for

terminates when it reaches a leaf node in the tree. In the example of Figure 3.1, the greedy-1 search procedure ultimately arrives at the leaf node $\mathbf{n}_{3,6}$.

Since the greedy-1 search is myopic, the search may lead to a leaf node $\mathbf{n}_{d-1,j}$ where the distance $d(\mathbf{n}_{d-1,j}, \mathbf{q})$ between the leaf node $\mathbf{n}_{d-1,j}$ and the feature descriptor $\mathbf{q}$ is high. A high-distance leaf node can reduce the retrieval accuracy when comparing tree histograms. To reduce the chances of selecting a high-distance leaf node, a more general greedy-r search can be employed [187]. In this case, the r most promising paths are explored simultaneously during the tree traversal. For example, Figure 3.1 shows greedy-3 search being applied in the same vocabulary tree as used in the previous greedy-1 search. Greedy-r search is defined formally in Algorithm 3.2. In this algorithm, the r-argmin operator selects the r inputs yielding the r smallest values for the cost function. When this algorithm is applied to the example of Figure 3.1 with $r = 3$, greedy-3 search leads to selection of $\mathbf{n}_{3,6}$, $\mathbf{n}_{3,11}$, and $\mathbf{n}_{3,18}$ as the three closest leaf nodes. From these three leaf nodes, we could then pick the closest leaf node if we apply hard binning or use all three leaf nodes if we apply soft binning, where the difference between the two binning options is explained in the next section.

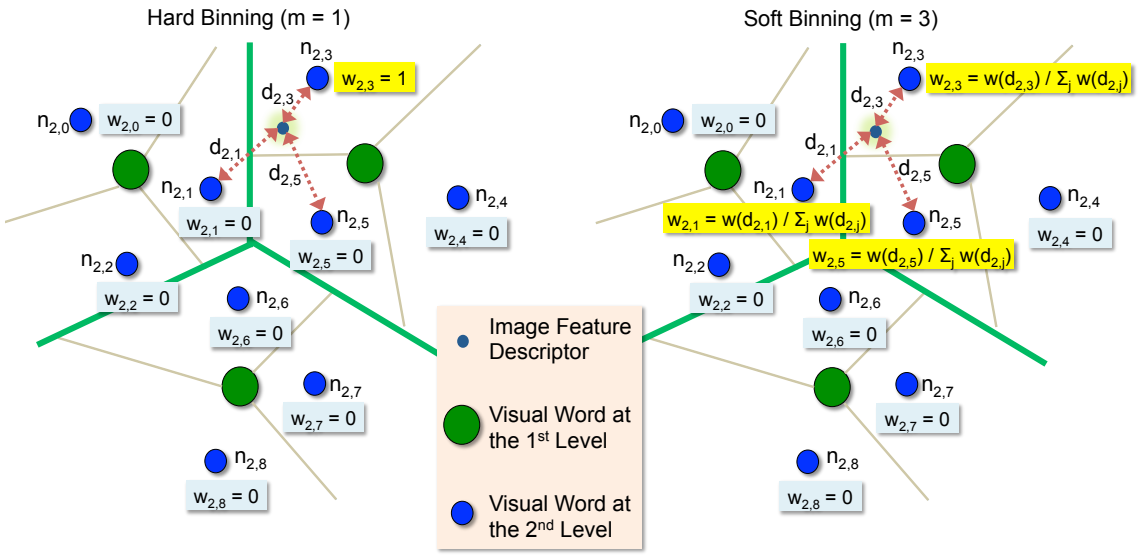


Figure 3.2: Voting for the m nearest leaf nodes in the vocabulary tree with hard binning ($m = 1$) and soft binning ($m = 3$).

3.1.2 Hard Binning versus Soft Binning

Since image feature descriptors are inherently noisy, two descriptors representing the same physical region in two different images may reach two different leaf nodes of the vocabulary tree during quantization. The hard binning method proposed in [158] selects the nearest leaf node as determined by the greedy search procedure described in Section 3.1.1, assigns a vote of 1 to that leaf node in the tree histogram, and assigns a vote of 0 to every other leaf node. An example is shown on the left half of Figure 3.2, where the leaf node $\mathbf{n}_{2,3}$ receives the entire vote of a nearby feature descriptor. Even though other leaf nodes such as $\mathbf{n}_{2,1}$ and $\mathbf{n}_{2,5}$ are also nearby, these nodes receive votes of 0 in the hard binning method.

To mitigate the effects of quantization errors, soft binning [172] enables m nearby leaf nodes to receive partial votes for a feature descriptor. This is illustrated on the right half of Figure 3.2 for $m = 3$, where the three closest leaf nodes $\mathbf{n}_{2,1}$, $\mathbf{n}_{2,3}$, and $\mathbf{n}_{2,5}$ receive fractional votes $w_{2,i} = w\left(d_{2,i}\right) / \sum_{j=1,3,5} w\left(d_{2,j}\right)$ for $i = 1, 3, 5$. Here, $d_{2,i}$ is the Euclidean distance between the feature descriptor $\mathbf{q}$ and the leaf node $\mathbf{n}_{2,i}$. The weighting function $w(x)$ is a nonnegative monotonically decreasing function for

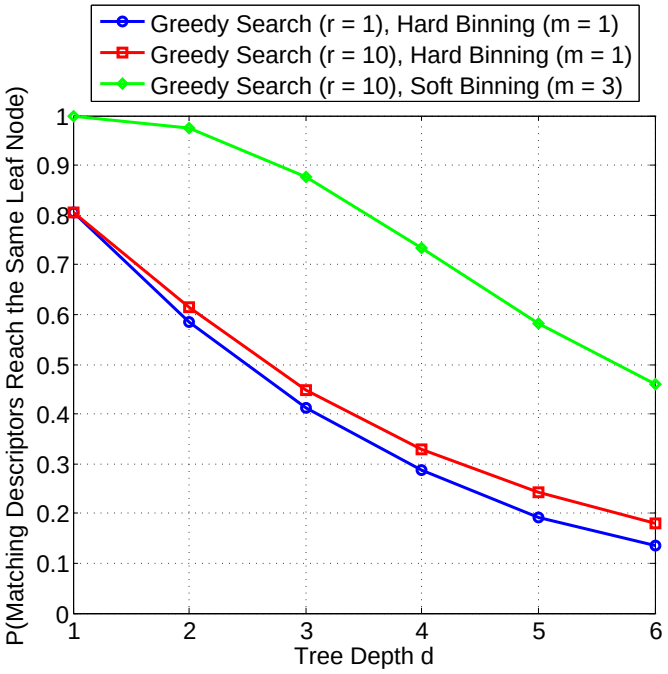


Figure 3.3: Probability that matching feature descriptors from two images reach the same leaf node in a vocabulary tree with branch factor $k = 10$ and varying depth d. Matching feature descriptors are found between all matching image pairs contained in the training dataset, as described in Appendix C. Each vocabulary tree is generated from the training feature descriptors, as described in Section 3.4.

$x \geq 0$, such as $w(x) = \exp\left(-x^2/\sigma^2\right)$ where σ is the standard deviation computed from a distribution of Euclidean distances between feature descriptors and their nearest leaf nodes.

The benefit of soft binning over hard binning can be seen if we consider the probability that two matching feature descriptors from two different images reach the same leaf node. Figure 3.3 plots this probability for vocabulary trees of branch factor $k = 10$ and varying depth $d = 1, \cdots, 6$. For harding binning ($m = 1$), we plot the probabilities for greedy-1 and greedy-10 search, where we see that greedy-10 offers some modest improvements, but the probability of reaching the same leaf node still drops to less than 0.2 when the depth increases to $d = 6$. Hence, a large majority of matching descriptors are quantized to different leaf nodes. For soft binning, each descriptor votes for m nearby leaf nodes, significantly increasing the chances that the

two descriptors will visit at least one leaf node in common. Figure 3.3 shows that soft binning with $m = 3$ coupled with greedy-10 search raises the probability to almost 0.5 at a depth of $d = 6$. The improvements of soft binning over hard binning are most important for large vocabulary trees. As we will show in Section 3.4, a moderate amount of soft binning significantly improves the retrieval accuracy compared to hard binning. The main costs of soft binning are an increase in the retrieval latency and an increase in the database memory usage.

3.1.3 Histogram Intersection

Suppose the feature descriptors of a query image and a database image are quantized through a vocabulary tree with greedy-r search and soft binning. Let $\mathbf{h}_q = \left[c^q_{d-1,0}, c^q_{d-1,1}, \cdots, c^q_{d-1,k^d-1}\right]$ and $\mathbf{h}_{db} = \left[c^{db}_{d-1,0}, c^{db}_{d-1,1}, \cdots, c^{db}_{d-1,k^d-1}\right]$ represent the counts of how often the k^d leaf nodes $\mathbf{n}_{d-1,0}$, $\mathbf{n}_{d-1,1}$, ..., $\mathbf{n}_{d-1,k^d-1}$ are visited by the query and database feature descriptors, respectively. Note that because we use soft binning, the counts are fractional numbers. Since the count for any interior node can be reconstructed from the leaf counts, the leaf histogram is sufficient to represent the entire tree histogram.

To compare two histograms, we use the term-frequency inverse-document-frequency (TF-IDF) histogram intersection method [182, 184]. The weighted histogram intersection score is given by

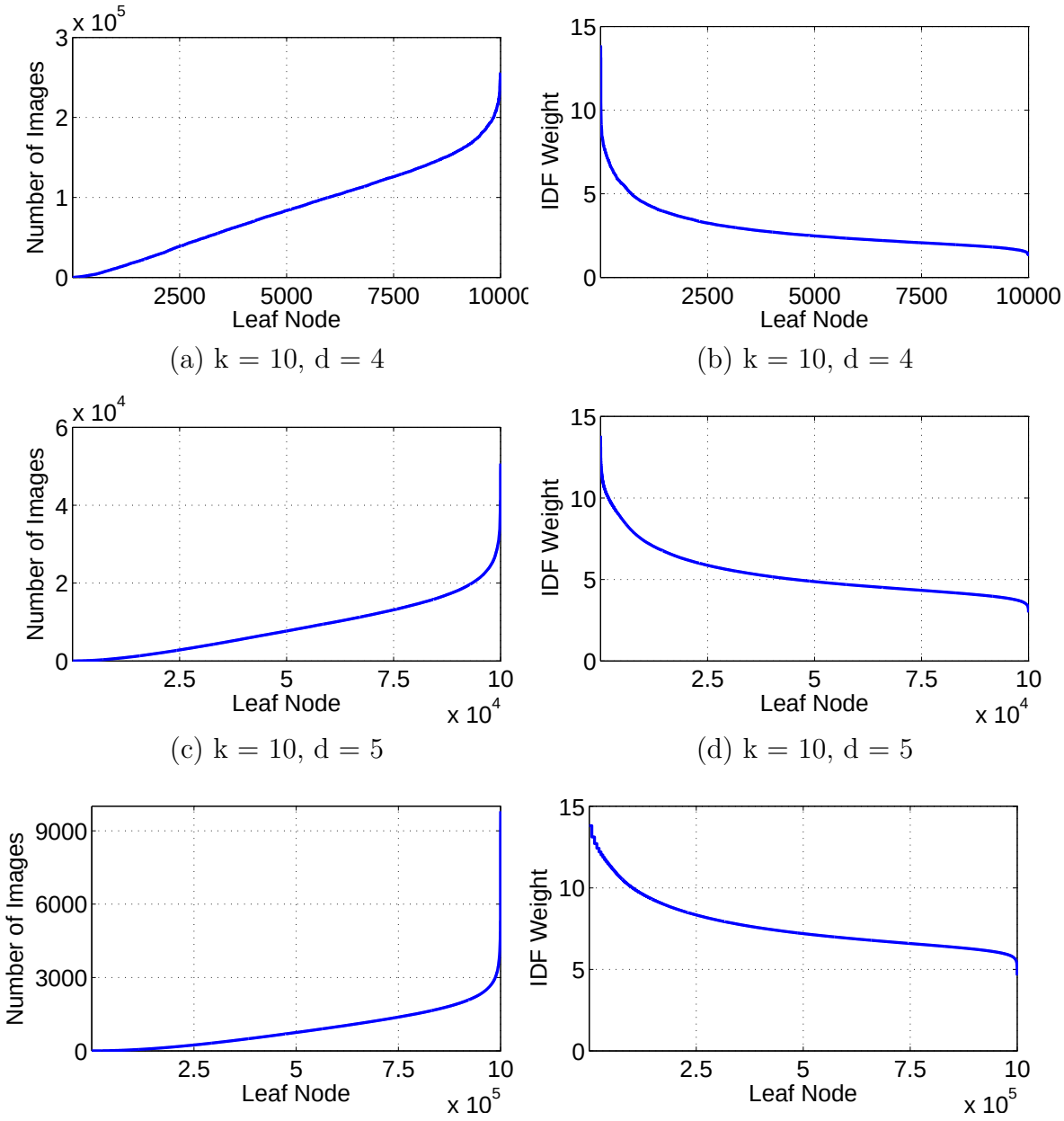
$$s\left(\mathbf{h}_q, \mathbf{h}_{db}\right) = \frac{\sum_{j=0}^{k^d-1} w_{\text{idf}}\left(\mathbf{n}_{d-1,j}\right) \min\left(c^q_{d-1,j}, c^{db}_{d-1,j}\right)}{\left(\sum_{j=0}^{k^d-1} w_{\text{idf}}\left(\mathbf{n}_{d-1,j}\right) c^q_{d-1,j}\right)\left(\sum_{j=0}^{k^d-1} w_{\text{idf}}\left(\mathbf{n}_{d-1,j}\right) c^{db}_{d-1,j}\right)}. \tag{3.1}$$

Here, $w_{\text{idf}}\left(\mathbf{n}_{d-1,j}\right)$ is the IDF weight for the node $\mathbf{n}_{d-1,j}$

$$w_{\text{idf}}\left(\mathbf{n}_{d-1,j}\right) = \log\left(\frac{N_{db}}{N_{d-1,j}}\right) \tag{3.2}$$

where N_{db} is the total number of images in the database and $N_{d-1,j}$ is the number of images with at least one feature descriptor visiting the leaf node $\mathbf{n}_{d-1,j}$.

The IDF weight favors the leaf nodes that are visited by fewer database images



(a) k = 10, d = 4 (b) k = 10, d = 4

(c) k = 10, d = 5 (d) k = 10, d = 5

(e) k = 10, d = 6 (f) k = 10, d = 6

Figure 3.4: (a, c, e) Number of images that visit each leaf node, out of a database of N_{db} = 1M images. (b, d, f) Corresponding IDF weights for the leaf nodes. The statistics are generated from the training dataset described in Appendix C.

because these nodes are more effective in differentiating between different database images during a query. Figure 3.4(a,c,e) plot the number of database images visiting each leaf node, out of a database of $N_{db} = 1\text{M}$ images contained in the training dataset described in Appendix C. In these plots, the leaf nodes are sorted by the ascending number of database images that visit the leaf node. The statistics are measured for three different vocabulary trees, all having branch factor $k = 10$ but with varying depth $d = 4, 5, 6$. The tree histograms are generated using soft binning with $m = 3$. Most images in the database have exactly $N_{\text{feat}} = 300$ local features chosen by a feature selection method that is optimized for image matching performance [76], but a small subset of images which contain small objects or texture-less surfaces have fewer than $N_{\text{feat}} = 300$ local features. Figure 3.4(b,d,f) plot the corresponding IDF weights calculated according to Equation 3.2. We can observe that the IDF weights can vary widely in value, meaning the different leaf nodes have widely varying amounts of discriminative power.

When computing the histogram intersection scores, the database normalization factor $N_{\text{idf}}(\mathbf{h}_{db}) = \sum_{j=0}^{k^d-1} w_{\text{idf}}(\mathbf{n}_{d-1,j})\, c_{d-1,j}^{db}$ can be pre-computed separately for each database image. The analogous query normalization factor is given by $N_{\text{idf}}(\mathbf{h}_q) = \sum_{j=0}^{k^d-1} w_{\text{idf}}(\mathbf{n}_{d-1,j})\, c_{d-1,j}^{q}$, which depends only on the features of the query image and will not affect the relative order of the database histogram intersection scores.

For a large vocabulary tree, each leaf histogram will be sparsely filled because only a small fraction of the leaf nodes are visited by an image's feature descriptors. For a leaf histogram $\mathbf{h} = \left[c_{d-1,0}, c_{d-1,1}, \cdots, c_{d-1,k^d-1}\right]$, denote the sequence of N_{pc} positive counts by $C(\mathbf{h}) = \left[c_{d-1,j_1}, c_{d-1,j_2}, \cdots, c_{d-1,j_{N_{pc}}}\right]$, where $N_{pc} \ll k^d$. Similarly, denote the sequence of node identifiers for these N_{pc} positive counts by $J(\mathbf{h}) = \left[j_1, j_2, \cdots, j_{N_{pc}}\right]$, where $j_1 < j_2 < \cdots < j_{N_{pc}}$. The histogram intersection score between a query histogram $\mathbf{h}_q$ and a database histogram $\mathbf{h}_{db}$ can be computed efficiently by Algorithm 3.3, which is adapted from an algorithm in information retrieval for intersecting two sparse posting lists [139]. The algorithmic complexity is approximately linear in the number of positive-count bins in a leaf histogram.

Algorithm 3.3 uses some simple operations to calculate the histogram intersection score: incrementing the pointers p_q and p_{db}, checking if $p_q \leq N_{pc}^q$ and $p_{db} \leq N_{pc}^{db}$, and

Algorithm 3.3 Histogram intersection between two sparsely filled histograms.

Require: Positive counts $C\left(\mathbf{h}_\theta\right) = \left[c^\theta_{d-1,j_1}, c^\theta_{d-1,j_2}, \cdots, c^\theta_{d-1,j_{N^\theta_{pc}}}\right]$ for $\theta \in \{q, db\}$

Require: Positive-count node identifiers $J\left(\mathbf{h}_\theta\right) = \left[j^\theta_1, j^\theta_2, \cdots, j^\theta_{N^\theta_{pc}}\right]$ for $\theta \in \{q, db\}$

$s \coloneqq 0$ ▷ Initialize intersection score to 0
$p_q \coloneqq 1$ ▷ Initialize pointer to first element in query list
$p_{db} \coloneqq 1$ ▷ Initialize pointer to first element in database list

while $p_q \leq N^q_{pc}$ and $p_{db} \leq N^{db}_{pc}$ **do**
  if $p_q = p_{db}$ **then**
    $s \coloneqq s + w_{\text{idf}}\left(\mathbf{n}_{d-1,j^q_{p_q}}\right) \min\left(c^q_{d-1,j_{p_q}}, c^{db}_{d-1,j_{p_{db}}}\right)$ ▷ Weighted intersection
    $p_q \coloneqq p_q + 1$
    $p_{db} \coloneqq p_{db} + 1$
  else if $j^q_{p_q} < j^{db}_{p_{db}}$ **then**
    $p_q \coloneqq p_q + 1$
  else
    $p_{db} \coloneqq p_{db} + 1$
  end if
end while

$s \coloneqq s / \left(N_{\text{idf}}\left(\mathbf{h}_q\right) \cdot N_{\text{idf}}\left(\mathbf{h}_{db}\right)\right)$ ▷ Normalize intersection score

checking if $j^q_{p_q} < j^{db}_{p_{db}}$. Although these operations are individually fast to perform, they can still lead to unnecessarily long delays when used repeatedly over a large database of images. If we are willing to reorganize the database tree histograms into an inverted index, an even more efficient method for evaluating the histogram intersection score is possible. We discuss this faster scoring method with the inverted index later in Section 3.3.

3.2 Tree Histogram Coding

To calculate the histogram intersection values given in Equation 3.1 between the query image and every database image, all of the N_{db} database tree histograms $\mathbf{h}_{db,1}$, ..., $\mathbf{h}_{db,N_{db}}$ must be stored in random access memory (RAM) for fast computations. Figure 3.5 plots the memory usage for the tree histograms of a database of approximately

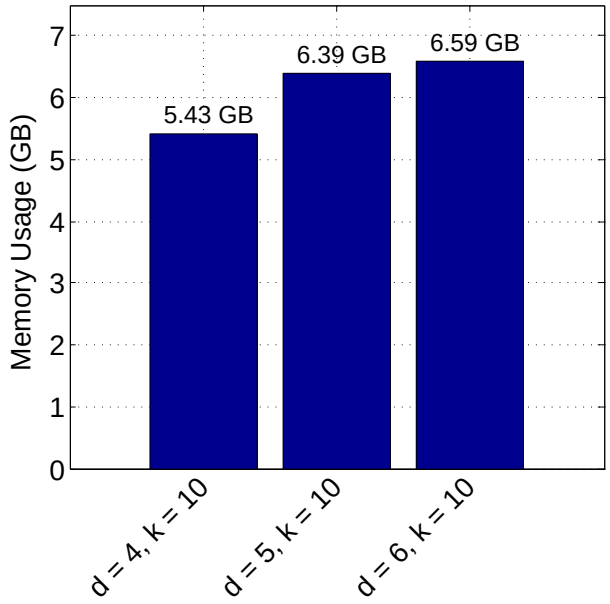


Figure 3.5: Memory usage of the N_{db} = 1,011,699 database tree histograms in the MPEG CDVS Dataset. The vocabulary trees have branch factor $k = 10$ and varying depth $d = 4, 5, 6$.

1M images contained in the MPEG CDVS Dataset, which is fully described in detail in Appendix A. Again, measurements are provided for three different vocabulary trees with branch factor $k = 10$ and varying depth $d = 4, 5, 6$. As explained before, soft binning with $m = 3$ is employed and each image has no more than $N_{\text{feat}} = 300$ local features. Since each leaf histogram has mostly empty bins, we store just the identifiers of the positive-count leaf bins as integer values and the fractional counts in those bins as floating point values. Even if we store just the information for the positive-count leaf bins, several gigabytes of RAM are required. This large memory footprint of tree histograms can (i) cause memory swapping on a server running other memory-intensive tasks and slow down the entire server, or (ii) prevent the vocabulary tree-based retrieval system from being deployed on a mobile device with a small memory capacity.

To reduce the memory usage of the tree histograms, we develop a method called tree histogram coding (THC). We first developed THC only for hard binning [38], but here we present a generalized version of THC that works for both hard and soft

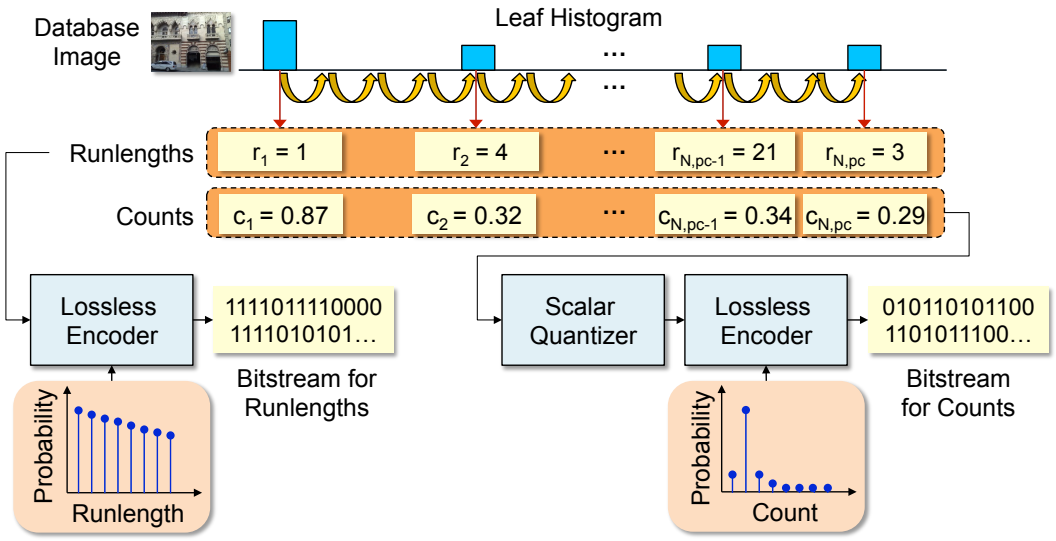


Figure 3.6: Usage of tree histogram coding (THC) to compress a database image's tree histogram into a pair of compact bitstreams representing the positive-count tree node identifiers and node counts.

binning. An overview of how THC compresses a database image's tree histogram is shown in Figure 3.6. The detailed discussion of THC is separated into two main parts: node identifier coding to be described in Section 3.2.1 and node count coding to be described in Section 3.2.1. After encoding each tree histogram, a pair of compressed bitstreams for the node identifiers and node counts are generated, which can be stored compactly in memory. Since the encoded tree histograms need to be decoded during a query to compute histogram intersection scores, having a fast decoder is very important for low-latency MVS applications. Hence, Section 3.2.3 describes how to build an extremely fast decoder with word-aligned codecs. Note that the tree histogram for each database image is encoded independently. Thus, when multiple CPU cores are available, the database tree histograms can be decoded simultaneously in parallel for faster access. In our previous work [38], we also recognized that the compressed tree histogram serves well as a compact global image signature that can be efficiently transmitted over a wireless network between a mobile device and a server, but we will not explore this application of THC any further in the current discussion.

3.2.1 Node Identifier Coding

As in Section 3.1.3, for a leaf histogram $\mathbf{h} = \left[c_{d-1,0}, c_{d-1,1}, \cdots, c_{d-1,k^d-1}\right]$, let the sequence of N_{pc} positive counts be denoted as $C(\mathbf{h}) = \left[c_{d-1,j_1}, c_{d-1,j_2}, \cdots, c_{d-1,j_{N_{pc}}}\right]$ and the corresponding sequence of positive-count node identifiers be denoted as $J(\mathbf{h}) = \left[j_1, j_2, \cdots, j_{N_{pc}}\right]$, where $j_1 < j_2 < \cdots < j_{N_{pc}}$. The sequence of positive-count node identifiers is losslessly compressed by a combination of runlength coding and entropy coding. We define the sequence of node runlengths as

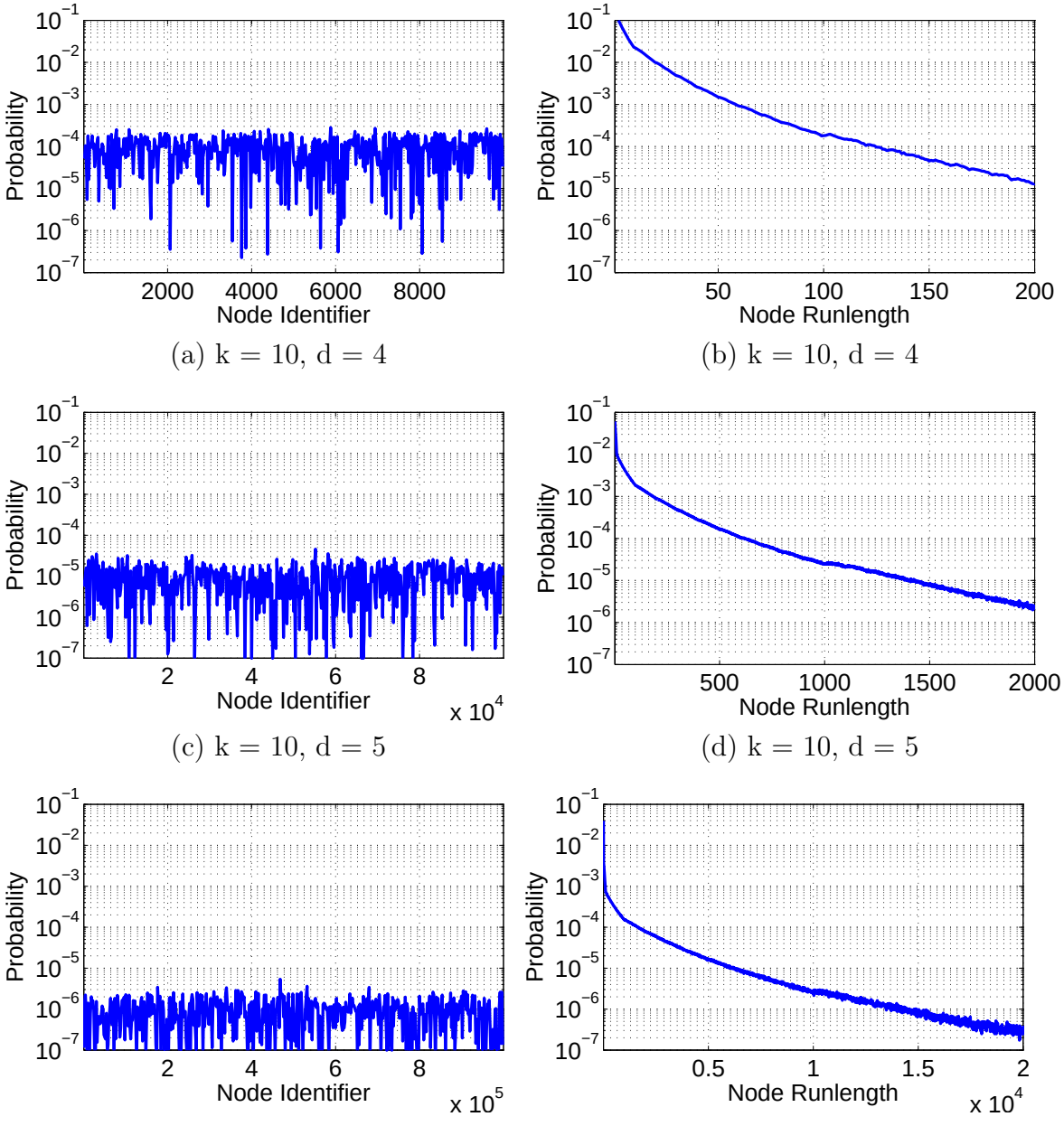
$$R(\mathbf{h}) = \left[r_1, r_2, \cdots, r_{N_{pc}}\right] = \left[j_1, j_2 - j_1, \cdots, j_{N_{pc}} - j_{N_{pc}-1}\right]. \tag{3.3}$$

Given the runlengths $R(\mathbf{h})$, the original identifiers $J(\mathbf{h})$ can be losslessly recovered:

$$\begin{aligned} j_1 &= r_1 \\ j_u &= \sum_{v=1}^{u} r_v \qquad u = 2, \cdots, N_{pc} \end{aligned} \tag{3.4}$$

Figure 3.7(a,c,e) plot the distributions of the node identifiers $J(\mathbf{h})$, generated from images in the training dataset described in Appendix C, for vocabulary trees of branch factor $k = 10$ and varying depth $d = 4, 5, 6$. Similarly, Figure 3.7(b,d,f) plot the distributions of the node runlengths $R(\mathbf{h})$. Again, soft binning with $m = 3$ and approximately $N_{\text{feat}} = 300$ local features per image are used. For each distribution $p(r)$, we compute the single-symbol entropy $H(p(r)) = -\sum_r p(r) \log_2 p(r)$ in bits and list the entropy value in Table 3.2. The runlength distributions are very peaky at low values, while the identifier distributions are more uniform over the range $\left\{1, \cdots, k^d\right\}$. Hence, the entropies $H(R(\mathbf{h}))$ of the runlengths are much lower than the entropies $H(J(\mathbf{h}))$ of the identifiers. This enables substantial memory savings when we utilize an entropy codec like an arithmetic codec [183] or the word-aligned codecs described in Section 3.2.3 to encode the sequence of runlengths. Note also from Table 3.2 that $H(J(\mathbf{h})) < \log_2 k^d$, because the leaf nodes are not visited uniformly often. Instead, the visit frequencies of the leaf nodes closely follow a Zipf distribution.

When we have approximately N_{feat} local features per image and we use soft binning



(a) k = 10, d = 4

(b) k = 10, d = 4

(c) k = 10, d = 5

(d) k = 10, d = 5

(e) k = 10, d = 6

(f) k = 10, d = 6

Figure 3.7: (a, c, e) Distributions of tree histogram node identifiers. (b, d, f) Distributions of tree histogram node runlengths. The statistics are generated from the training dataset described in Appendix C.

Tree	$H_J = H(J(\mathbf{h}))$	$H_R = H(R(\mathbf{h}))$	$H_J - H_R$	$G(mN_{\text{feat}})$	$\log_2 k^d$
$k = 10$, $d = 4$	12.94	4.80	8.14	8.38	13.29
$k = 10$, $d = 5$	16.14	7.88	8.26	8.38	16.61
$k = 10$, $d = 6$	19.48	11.20	8.28	8.38	19.93

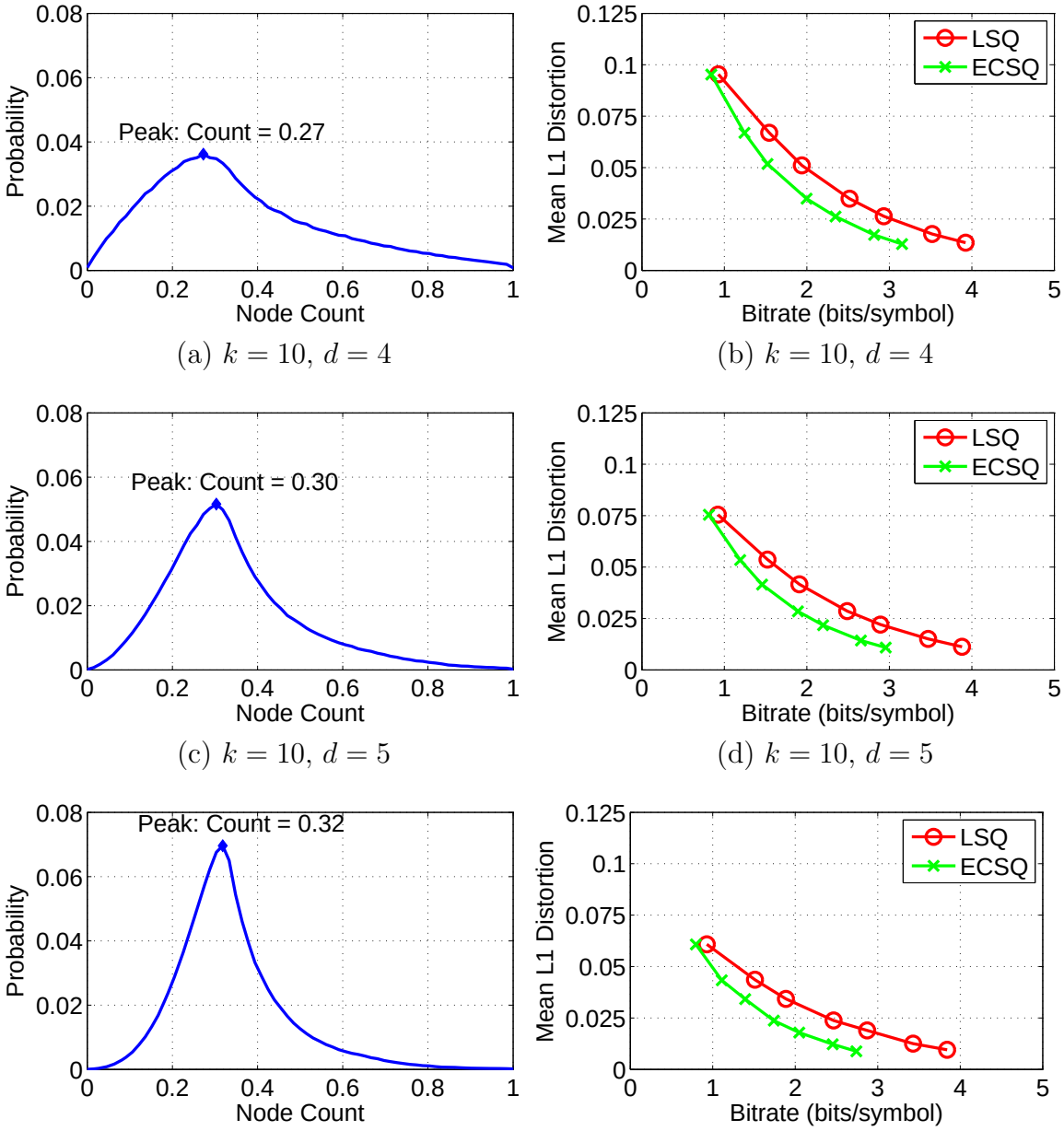
Table 3.2: Entropy values in bits for tree histogram node identifiers and node runlengths. Each image has approximately $N_{\text{feat}} = 300$ local features, and soft binning with $m = 3$ is applied during quantization. The statistics are generated from the training dataset described in Appendix C.

with m leaf nodes per feature, there are approximately mN_{feat} different leaf nodes visited. The order in which these mN_{feat} leaf nodes are visited is not important for image retrieval. Since there are $(mN_{\text{feat}})!$ possible orderings among the mN_{feat} leaf nodes, we can expect to save $\log_2(mN_{\text{feat}})!$ bits for the entire set of node identifiers if we ignore this ordering information. The memory saved per symbol by encoding node runlengths instead of directly encoding node identifiers is close to the ordering gain $G(mN_{\text{feat}}) = \frac{1}{mN_{\text{feat}}} \log_2(mN_{\text{feat}})!$ bits/symbol, as shown in Table 3.2.

3.2.2 Node Count Coding

Since changing the leaf node identifier by even a small value can significantly degrade the retrieval accuracy, we pursued lossless coding of the node identifiers in the previous section. In contrast, even a coarse quantization of the node counts has a very mild effect on retrieval accuracy, possibly even improving the retrieval accuracy slightly after quantization as we show in Section 3.4. The sequence of positive node counts $C(\mathbf{h}) = \left[c_{d-1,j_1}, \cdots, c_{d-1,j_{N_{pc}}}\right]$ is compressed by a combination of scalar quantization and entropy coding.

Figure 3.8(a,c,e) plot the distribution $p(c)$ of the node counts for vocabulary trees of branch factor $k = 10$ and varying depth $d = 4, 5, 6$. As before, each image uses approximately $N_{\text{feat}} = 300$ local features and soft binning uses $m = 3$. For each tree, the peak of the count distribution $p(c)$ occurs close to a count value of $c = 0.3$, and the probability falls to 0 for very small or very large count values. To effectively quantize the counts, we design two different scalar quantizers and compare their performances:



(a) $k = 10$, $d = 4$ (b) $k = 10$, $d = 4$

(c) $k = 10$, $d = 5$ (d) $k = 10$, $d = 5$

(e) $k = 10$, $d = 6$ (f) $k = 10$, $d = 6$

Figure 3.8: (a, c, e) Distributions of tree histogram node counts. (b, d, f) Distortion-rate trade-offs for Lloyd scalar quantization (LSQ) and entropy-constrained scalar quantization (ECSQ) of the node counts. The statistics are generated from the training dataset described in Appendix C.

1. First, we perform Lloyd scalar quantization(LSQ) [131, 132]. LSQ is the optimal quantizer when fixed-length coding is used to encode the quantization indices. Sometimes, fixed-length coding may be preferred over entropy coding for greater computational simplicity, if some loss in coding efficiency is acceptable. Since the L_1 distortion has a close connection to the histogram intersection score defined in Section 3.1.3, we optimize the quantizer to minimize the mean L_1 distortion between the original and quantized counts. Suppose the quantizer uses N_c centroids. Then, LSQ determines the centroids $\{\hat{c}_i\}_{i=1}^{N_c}$ which minimize the mean L_1 distortion D after quantization:

$$\text{minimize} \quad D = \sum_{i=1}^{N_c} E\left[|c - \hat{c}_i| \;\middle|\; c \in C_i\right] P\left(c \in C_i\right) \tag{3.5}$$

$$C_i = \{c \;:\; q(c) = i\} \tag{3.6}$$

   where $q(c) = i$ means the count c has been quantized to the i^{th} centroid. For LSQ, it is optimal to use a nearest-neighbor assignment rule: $q(c) = \text{argmin}_i \, |c - \hat{c}_i|$. We generate the quantizer centroids using 500 iterations of the Lloyd algorithm.

2. Second, we perform entropy-constrained scalar quantization (ECSQ) [69, 47]. This is the optimal quantizer when entropy coding is used to encode the quantization indices. We use ECSQ to minimize the mean L_1 distortion D subject to the constraint that the entropy $H\left(p(\hat{c})\right)$ of the quantized counts must be less than a desired bitrate R.

$$\text{minimize} \qquad D = \sum_{i=1}^{N_c} E\left[|c - \hat{c}_i| \;\middle|\; c \in C_i\right] P\left(c \in C_i\right) \tag{3.7}$$

$$\text{s.t.} \qquad H\left(p(\hat{c})\right) = -\sum_{i=1}^{N_c} P\left(c \in C_i\right) \log_2 P\left(c \in C_i\right) \;\leq\; R \tag{3.8}$$

   where as before $C_i = \{c \;:\; q(c) = i\}$. For ECSQ, a nearest-neighbor assignment rule for $q(c)$ is generally sub-optimal. To determine the quantizer parameters,

we use the algorithm in [47], which employs the technique of Lagrange multipliers to solve an unconstrained optimization problem. In turn, the constrained optimization problem in Equations 3.7-3.8 is also solved when the correct value of the Lagrange multiplier λ is chosen. We search for a value of λ that enables the entropy $H\left(p(\hat{c})\right)$ of the quantized counts to match the target bitrate R. For each sampled value of λ during the search process, we apply 500 iterations of a generalized Lloyd algorithm to optimize the quantizer parameters. Although ECSQ has a moderately more complicated offline training phase than LSQ, both methods have the same computational efficiency for quantizing database and query feature counts.

Figure 3.8(b,d,f) plots the distortion-rate trade-offs of the node counts using LSQ and ECSQ. For the bitrate, we report the entropy of the quantization indices. Since ECSQ minimizes the mean distortion under a rate constraint, it is expected to have lower distortion than other quantizers including LSQ at any given bitrate. This is confirmed experimentally in Figure 3.8(b,d,f) for all three vocabulary trees. Thus, for all subsequent experiments, we will use ECSQ to quantize the feature counts. As the vocabulary tree size increases, the mean L_1 distortion decreases at the same bitrate, because the node count distribution becomes more peaky.

During histogram intersection, the counts are multiplied by IDF weights, as shown in Equation 3.1. As before, let the sequence of positive-count node identifiers for a histogram $\mathbf{h}$ be denoted as $J\left(\mathbf{h}\right) = \left[j_1, j_2, \cdots, j_{N_{pc}}\right]$. When the IDF weighting is taken into account, the weighted L_1 distance between the original counts $C\left(\mathbf{h}\right) = \left[c_{d-1,j_1}, \cdots, c_{d-1,j_{N_{pc}}}\right]$ and the quantized counts $\widehat{C}\left(\mathbf{h}\right) = \left[\hat{c}_{d-1,j_1}, \cdots, \hat{c}_{d-1,j_{N_{pc}}}\right]$ is calculated as

$$\left\|C\left(\mathbf{h}\right) - \hat{C}\left(\mathbf{h}\right)\right\|_{\text{idf}} \approx \frac{1}{N_{\text{idf}}\left(\mathbf{h}\right)} \sum_{j \in J(\mathbf{h})} w_{\text{idf}}\left(\mathbf{n}_{d-1,j}\right) \left|c_{d-1,j} - \hat{c}_{d-1,j}\right| \tag{3.9}$$

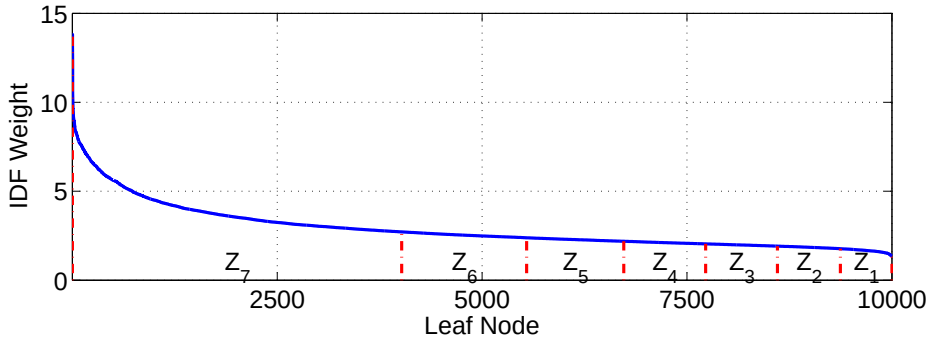
assuming the quantization of the counts does not significantly change the normalization factor $N_{\text{idf}}\left(\mathbf{h}\right)$. From Equation 3.9, we can see that quantization errors for different nodes are weighted differently. Hence, quantization should be coarser for nodes with small IDF weights than for nodes with large IDF weights.

Tree	H (quantized node count)
$k = 10$, $d = 4$	1.99
$k = 10$, $d = 5$	1.88
$k = 10$, $d = 6$	1.75

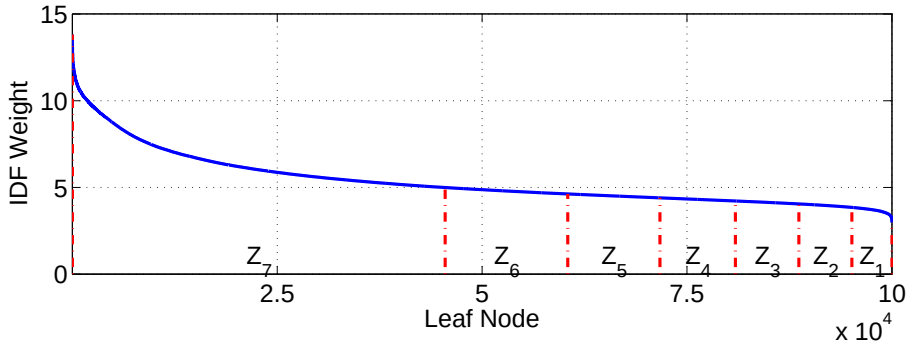
Table 3.3: Entropy values in bits for quantized node counts. Each image has approximately $N_{\text{feat}} = 300$ local features, and soft binning with $m = 3$ is applied. The statistics are generated from the training dataset described in Appendix C.

Designing a different quantizer for each possible IDF weight would lead to a large number of quantizers and be too computationally expensive to implement in a practical MVS system. Instead, we seek an efficient solution where we divide the IDF weights into N_{zones} different zones $Z_1, \cdots, Z_{N_{\text{zones}}}$. Figure 3.9 shows an example for $N_{\text{zones}} = 7$ zones. These N_{zones} zones correspond to N_{zones} different distortion-rate points $\{(R_n, D_n)\}_{n=1}^{N_{\text{zones}}}$ in the ECSQ distortion-rate trade-off, where $R_1 < R_2 < \cdots < R_{N_{\text{zones}}}$. For example, Z_1 is assigned the coarsest quantizer with the lowest bitrate R_1, whereas $Z_{N_{\text{zones}}}$ is assigned the finest quantizer with the highest bitrate $R_{N_{\text{zones}}}$. The N_{zones} zones are designed to allocate the same number of counts throughout the database within each zone. Therefore, the average bitrate and distortion are $\bar{R} = \frac{1}{N_{\text{zones}}} \sum_{n=1}^{N_{\text{zones}}} R_n$ and $\bar{D} = \frac{1}{N_{\text{zones}}} \sum_{n=1}^{N_{\text{zones}}} D_n$, respectively.

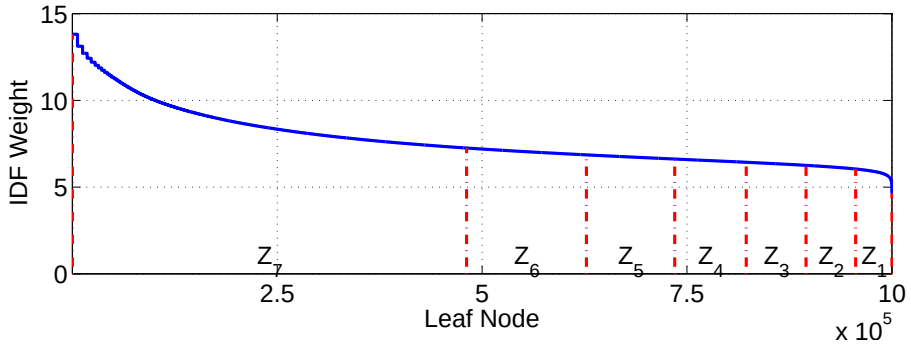
The quantization indices are losslessly encoded using an entropy codec like an arithmetic codec [183] or the word-aligned codecs of Section 3.2.3. Table 3.3 lists the entropies of the quantized counts for the three different vocabulary trees of varying depth. Whereas the entropy of the node identifier runlengths changes significantly as the tree size increases (see Table 3.2), the entropy of the quantized node counts changes much less dramatically with tree size. Note also that the entropy of the quantized node counts is much smaller than the entropy of the node identifier runlengths. Particularly for a large tree, e.g., $k = 10$ and $d = 6$, the memory usage of the compressed node counts is then substantially less than the memory usage of the compressed node identifiers.



(a) $k = 10$, $d = 4$



(b) $k = 10$, $d = 5$



(c) $k = 10$, $d = 6$

Figure 3.9: Different quantization zones for the leaf nodes based on the IDF weights. The IDF weights are the same as those shown in Figure 3.4.

3.2.3 Word-Aligned Codecs

Having fast decoding capabilities is very important for low-latency MVS applications. An arithmetic codec [183] provides excellent compression gains that approach

Selector	Data
4 bits	28 bits
A	28 codewords, each 1 bit, representing values in {0, 1}
B	14 codewords, each 2 bits, representing values in {0, ..., 3}
C	9 codewords, each 3 bits, representing values in {0, ..., 7}
D	7 codewords, each 4 bits, representing values in {0, ..., 15}
E	5 codewords, each 5 bits, representing values in {0, ..., 31}
F	4 codewords, each 7 bits, representing values in {0, ..., 127}
G	3 codewords, each 9 bits, representing values in {0, ..., 511}
H	2 codewords, each 14 bits, representing values in {0, ..., 16383}
I	1 codeword, each 28 bits, representing values in {0, ..., 268435455}

Figure 3.10: Division of a 32-bit computer word into a 4-bit selector portion and a 28-bit data portion. For each selector symbol (A, ..., I), the carryover codec assigns a different meaning to the bits in the data portion.

the entropies of the node runlengths and quantized node counts. The arithmetic codec, however, requires a relatively long decoding time, because the codec often uses multiple reads from memory to decode a single symbol. Much faster decoding can be achieved with word-aligned codecs [8, 149]. Word-aligned codecs enforce the constraint that a codeword is fully contained within a 32-bit computer word and never stretches across the boundary of two different 32-bit computer words. Whereas traditional entropy coding methods try to minimize the length of each codeword, word-aligned codecs try to maximize the number of codewords that can be packed into a 32-bit computer word. These two goals seem similar and indeed lead to comparable compression gains, but the effects on the decoding speed are dramatic because of the different ways in which the computer memory is accessed during decoding.

The carryover codec [8] is a word-aligned codec built on the idea of dividing each 32-bit word into a selector portion and a data portion, as illustrated in Figure 3.10. The selector portion uses 4 bits, which means up to 16 different modes can be signaled to describe the format of the following 28-bit data portion. Let $b(x, s)$ be the binary encoding of an integer x with s bits of precision. For example, $b(1, 1) = 1$, $b(1, 2) = 01$, $b(1, 3) = 001$, and so on. Now, suppose the next 28 data symbols $x_1, \cdots, x_{28}$ to be encoded are all 0 or 1. Then, the selector would be set to A and the 28 data bits

would be $b(x_1, 1), \cdots, b(x_{28}, 1)$, a sequence of 28 different 1-bit codewords indicating whether each of the 28 data symbols is 0 or 1. This is the best possible scenario for compression efficiency. The second best possible scenario happens if the next 14 data symbols $x_1, \cdots, x_{14}$ are all in the set $\{0, 1, 2, 3\}$. In this case, the selector would be set to B and the 28 data bits would be $b(x_1, 2), \cdots, b(x_{14}, 2)$, a sequence of 14 different 2-bit codewords representing numbers in $\{0, 1, 2, 3\}$. As we go from the selector A to the selector I, compression efficiency decreases to accommodate the less frequently occurring data symbols that require higher-precision encoding, analogous to the mechanism of using longer codeword lengths to represent less probable symbols in traditional entropy coding.

The carryover codec automatically adapts to the current data statistics and selects the most memory-efficient selector/data combination based on the data symbols encountered. In Section 3.2.1-3.2.2, we saw that the node runlengths had a high likelihood of attaining small positive values and the node count quantization indices are nonuniformly distributed over the set $\{1, \cdots, 8\}$. The carryover codec's design is well suited to the natural statistics of the node runlengths and node count quantization indices, and thus it will provide good compression performance for these data sources. Because of its simplicity, the carryover codec offers very fast decoding functionality and only experiences a minor loss in compression gain compared to a far more complex codec like the arithmetic codec.

A more sophisticated word-aligned codec that also has fast decoding capabilities is the recursive bottom up complete (RBUC) codec [149]. Figure 3.11 shows a block diagram for recursively encoding a sequence of positive integers using the RBUC codec. First, given an input sequence $\mathbf{x} = [x_1, \cdots, x_N]$ where N is an even number, the selector symbols are computed:

$$[s_1, \cdots, s_N] = [\lceil \log(x_1 + 1) \rceil, \cdots, \lceil \log(x_N + 1) \rceil]. \tag{3.10}$$

Next, the selector symbols are grouped by taking the max amongst every consecutive

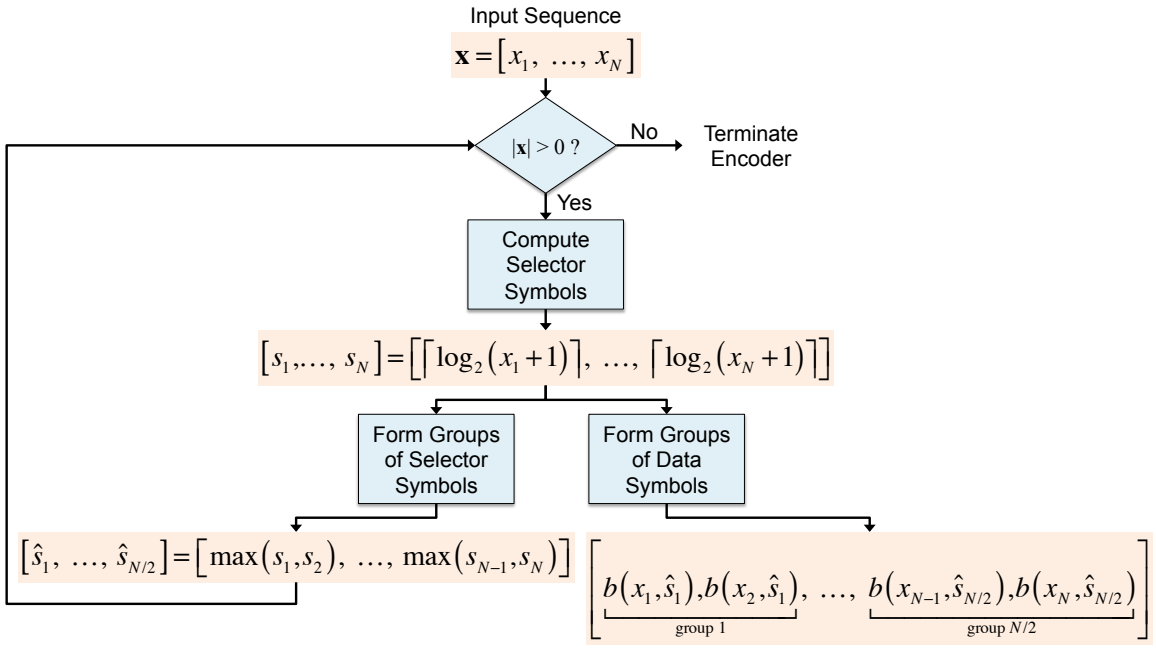


Figure 3.11: Block diagram for encoding a sequence of positive integers using the recursive bottom up complete (RBUC) codec.

pair of selector symbols:

$$\left[\hat{s}_1, \cdots, \hat{s}_{N/2}\right] = \left[\max\left(s_1, s_2\right), \cdots, \max\left(s_{N-1}, s_N\right)\right]. \tag{3.11}$$

Based on the grouped selector symbols, the data bits are determined:

$$\text{data} = \left[b\left(x_1, \hat{s}_1\right), b\left(x_2, \hat{s}_1\right), \cdots, b\left(x_{N-1}, \hat{s}_{N/2}\right), b\left(x_N, \hat{s}_{N/2}\right)\right] \tag{3.12}$$

where consecutive pairs of data symbols are encoded with the same precision. Given the grouped selector symbols $\left[\hat{s}_1, \cdots, \hat{s}_{N/2}\right]$, we can determine how many bits to allocate for each data symbol and subsequently pack as many codewords as possible into each 32-bit computer word. As the final step in RBUC encoding, the grouped selector symbols $\left[\hat{s}_1, \cdots, \hat{s}_{N/2}\right]$ are regarded as a new input sequence and recursively encoded using the same procedure. Because the length of the input sequence is cut in half during each recursion, about $\log_2 N$ recursions are required to complete RBUC

encoding of the input sequence $\mathbf{x} = [x_1, \cdots, x_N]$.

Like the carryover codec, the RBUC codec uses selector symbols and data symbols to adaptively encode an input sequence while obeying the computer word boundary constraints. The RBUC codec has a more versatile mechanism to exploit the changing statistics of both selector and data symbols. Hence, the RBUC codec provides a greater compression gain than the carryover codec, at the expense of a slightly longer decoding time. The decoding delay of the RBUC codec, however, is still much lower than that of the arithmetic codec, so the RBUC codec is also an excellent choice for low-latency MVS applications. In Section 3.4, we systematically compare the performance of the arithmetic codec, carryover codec, and RBUC codec in encoding the node runlengths and quantized node counts for a large image database.

3.3 Inverted Index Coding

Section 3.1.3 described how to compute the histogram intersection score between the query leaf histogram and every database leaf histogram. For a large database, e.g., $N_{db} = 1\text{M}$ images, evaluation of Equation 3.1 by Algorithm 3.3 for every database image can take a long time. Although Algorithm 3.3 exploits the fact that most of the terms in the sum $\sum_{j=0}^{k^d-1} w_{\text{idf}}(\mathbf{n}_{d-1,j}) \min\left(c^q_{d-1,j}, c^{db}_{d-1,j}\right)$ of Equation 3.1 are zero and hence do not affect the histogram intersection score, the algorithm still requires a large number of operations to update two positional pointers and to check the relative values of these pointers.

The inverted index [153, 229, 228] avoids the extraneous calculations in Algorithm 3.3 by storing a list of which database images have positive counts for each leaf node. Figure 3.12 shows a portion of the inverted index for a vocabulary tree. Only the inverted lists $\mathbf{i}_0$ and $\mathbf{i}_1$ for the leaf nodes $\mathbf{n}_{d-1,0}$ and $\mathbf{n}_{d-1,1}$ are shown, but every other leaf node also has its own inverted list. The inverted list $\mathbf{i}_u$ for leaf node $\mathbf{n}_{d-1,u}$ consists of a sequence of M_u image identifiers $J(\mathbf{i}_u) = [j_{u,1}, \cdots, j_{u,M_u}]$ and a corresponding sequence of M_u image counts $C(\mathbf{i}_u) = [c_{u,1}, \cdots, c_{u,M_u}]$. To compute histogram intersection scores with an inverted index, we can use Algorithm 3.4.

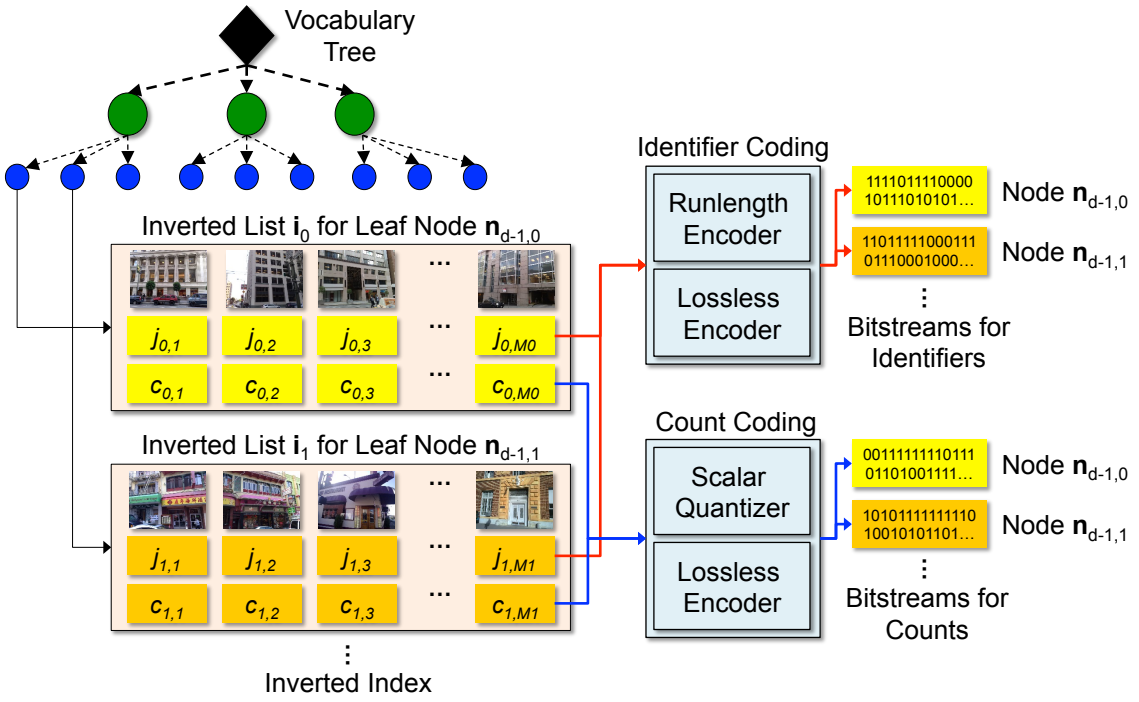


Figure 3.12: Illustration of inverted index coding (IIC) for the inverted lists of two leaf nodes in the vocabulary tree.

Although the steps used in this algorithm are similar to the steps used in Algorithm 3.3, now we can avoid the additional pointer arithmetic operations of Algorithm 3.3, and we visit only the inverted lists for the leaf nodes reached by the local features of the query image. At the end, we additionally normalize the n^{th} database score by the factor $N_{\text{idf}}(\mathbf{h}_{db,n}) = \sum_{j=0}^{k^d-1} w_{\text{idf}}(\mathbf{n}_{d-1,j})\, c_{d-1,j}^{db,n}$ and optionally by the factor $N_{\text{idf}}(\mathbf{h}_q) = \sum_{j=0}^{k^d-1} w_{\text{idf}}(\mathbf{n}_{d-1,j})\, c_{d-1,j}^{q}$.

Whereas the tree histograms contained node identifiers and node counts, the inverted index contains image identifiers and image counts. In RAM, the image identifiers are stored as integer values and the image counts generated from soft binning are stored as floating point values. Since there is a one-to-one mapping from the symbols in the inverted lists to the symbols in the database tree histograms, the memory usage of the inverted index is identical to the memory usage of the database tree histograms plotted in Figure 3.5. Hence, the inverted index also suffers from the problem of a large memory footprint when the database contains a large number of images.

Algorithm 3.4 Histogram intersection calculation with an inverted index.

Require: Image counts $C(\mathbf{i}_u) = [c_{u,1}, \cdots, c_{u,M_u}]$ for $u = 0, \cdots, k^d - 1$
Require: Image identifiers $J(\mathbf{i}_u) = [j_{u,1}, \cdots, j_{u,M_u}]$ for $u = 0, \cdots, k^d - 1$

$s_n \coloneqq 0$ for $n = 1, \cdots, N_{db}$ ▷ Initialize database image scores
for $u = 0, \cdots, k^d - 1$ **do**
  if $c^q_{d-1,u} > 0$ **then** ▷ Traverse inverted lists for nodes visited by query image
    for $v = 1, \cdots, M_u$ **do** ▷ Only M_u images in the current inverted list
      $s_{j_{u,v}} \coloneqq s_{j_{u,v}} + w_{\text{idf}}(\mathbf{n}_{d-1,u}) \min\left(c^q_{d-1,u}, c^{db,j_{u,v}}_{d-1,u}\right)$ ▷ Weighted intersection
    end for
  end if
end for

for $n = 1, \cdots, N_{db}$ **do**
  $s_n \coloneqq s_n / \left(N_{\text{idf}}(\mathbf{h}_q) \cdot N_{\text{idf}}(\mathbf{h}_{db,n})\right)$ ▷ Normalize intersection scores
end for

We develop a method called inverted index coding (IIC) for substantially reducing the memory usage of the inverted index, while preserving the index's advantage of facilitating fast score computations. The process of compressing inverted lists into compact bitstreams using IIC is shown in Figure 3.12. Each inverted list is independently encoded into separate bitstreams, so that the compressed inverted lists can be independently decoded during a query. This is important because Algorithm 3.4 only requires the inverted lists for the few hundred leaf nodes visited by a query image, so we decode only this small set of inverted lists. This enables inverted index decoding to be a very fast process. When multiple CPU cores are available, independent decoding of the inverted lists also allows for fast parallel decoding of multiple lists simultaneously. IIC is carried out in two main parts: (i) image identifier coding as described in Section 3.3.1 and (ii) image count coding as described in Section 3.3.2.

3.3.1 Image Identifier Coding

Given a sorted sequence of M_u image identifiers $J(\mathbf{i}_u) = [j_{u,1}, \cdots, j_{u,M_u}]$ where $j_{u,1} < j_{u,2} < \cdots < j_{u,M_u}$, we define a sequence of image runlengths:

$$R(\mathbf{i}_u) = [r_{u,1}, r_{u,2}, \cdots, r_{u,M_u}] = [j_{u,1}, j_{u,2} - j_{u,2}, \cdots, j_{u,M_u} - j_{u,M_u-1}]. \tag{3.13}$$

Tree	$H_J = H(J(\mathbf{i}))$	$H_R = H(R(\mathbf{i}))$	$H_J - H_R$	$E[G(M_u)]$	$\log_2 N_{db}$
$k = 10$, $d = 4$	19.94	5.67	14.27	14.29	19.95
$k = 10$, $d = 5$	19.94	9.78	10.16	10.28	19.95
$k = 10$, $d = 6$	19.95	12.63	7.32	7.42	19.95

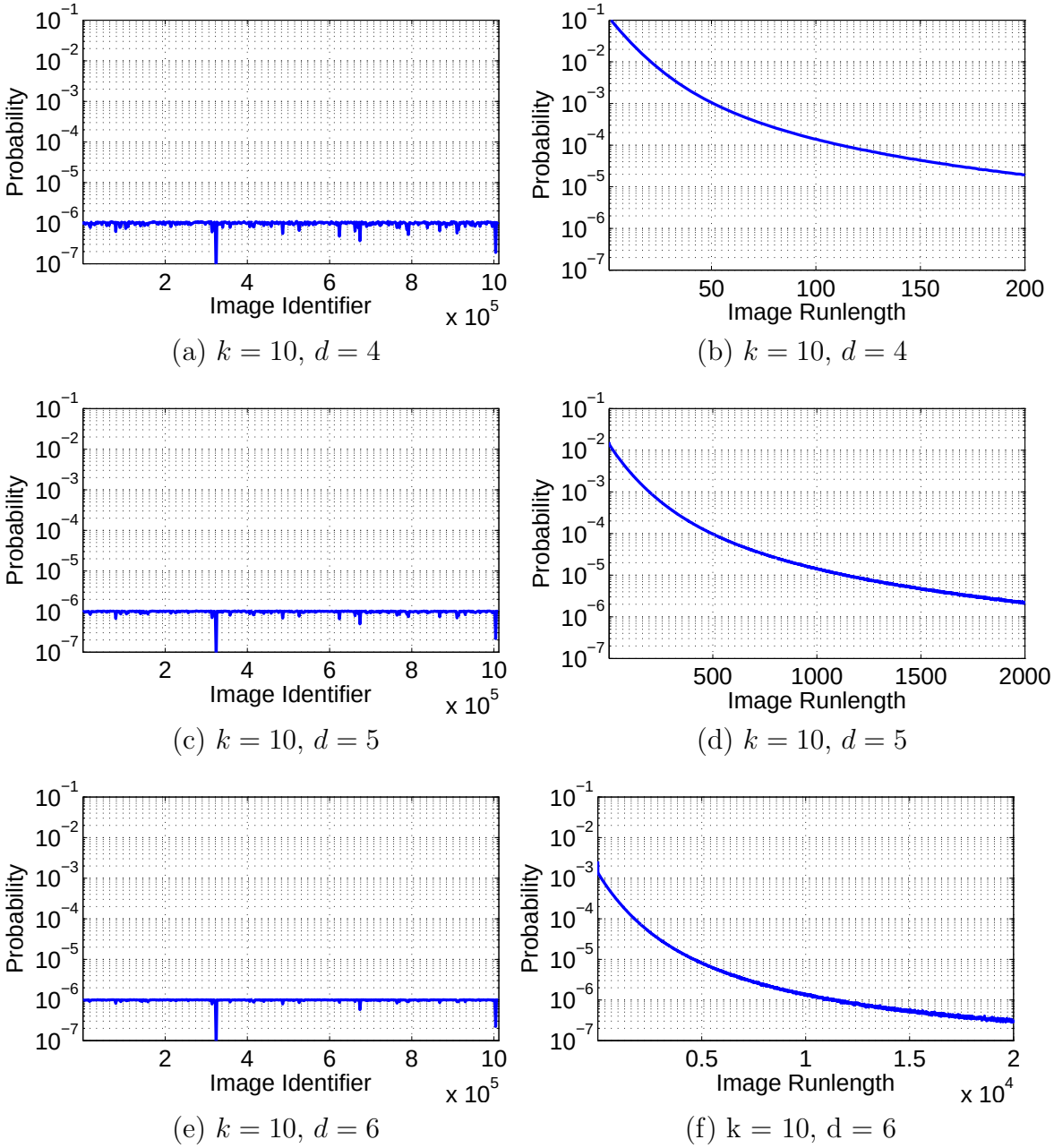
Table 3.4: Entropy values in bits for inverted index image identifiers and image runlengths. Each image has approximately $N_{\text{feat}} = 300$ local features, and soft binning with $m = 3$ is applied. The statistics are generated from the training dataset described in Appendix C.

This operation is similar in spirit to forming the node runlengths in Section 3.2.1, but the statistics of the image runlengths are quite different from the statistics of the node runlengths.

Figure 3.13(a,c,e) plot the distributions of the image identifiers for a database of $N_{db} \approx 1$M images contained in the training dataset of Appendix C. Vocabulary trees of branch factor $k = 10$ and depth $d = 4, 5, 6$, and soft binning with $m = 3$ are used. As before, each image has approximately $N_{\text{feat}} = 300$ local features. The image identifier distributions are close to uniform over the range $\{1, \cdots, N_{db}\}$. Therefore, the image identifiers have entropies close to $\log_2 N_{db}$ for all three vocabulary trees, as shown in Table 3.4. The image identifier distributions are not exactly uniform for two reasons: (i) some database images have fewer than $N_{\text{feat}} = 300$ features, due to a lack of textured surfaces, and (ii) some database images have repetitive structures, leading to many features being quantized to the same leaf nodes.

In contrast, Figure 3.13(b,d,f) plot the distributions of the image runlengths. These distributions decay rapidly as the runlength value increases, allowing the runlengths to have much lower entropies than the identifiers, analogous to the effect we observed previously for node runlengths versus node identifiers. The image runlengths can subsequently be compressed efficiently using an entropy codec like the arithmetic codec, carryover codec, or RBUC codec.

The memory savings in encoding image runlengths instead of image identifiers can again be interpreted as the amount of information conveyed by the ordering among a set of symbols, but now a more detailed analysis is required to precisely quantify the exact savings. In Section 3.2.1, the number of symbols in each tree histogram



(a) $k = 10$, $d = 4$ (b) $k = 10$, $d = 4$

(c) $k = 10$, $d = 5$ (d) $k = 10$, $d = 5$

(e) $k = 10$, $d = 6$ (f) k = 10, d = 6

Figure 3.13: (a, c, e) Distributions of inverted index image identifiers. (b, d, f) Distributions of inverted index image runlengths. The statistics are generated from the training dataset described in Appendix C.

is approximately constant: mN_{feat}. The number of symbols M_u in an inverted list, however, varies widely from list to list. This causes the ordering gain $G(M_u)$ to vary widely for different inverted lists. We can compute the mean ordering gain $E[G(M_u)]$ as follows. As we mentioned previously, the probability that a leaf node is visited by a feature descriptor can be modeled closely by a Zipf distribution:

$$p_{\text{feat}}(u) = \frac{1/(u+1)^s}{\sum_{v=0}^{k^d-1} 1/(v+1)^s} \qquad u = 0, \cdots, k^d - 1 \tag{3.14}$$

where the Zipf parameter s can be accurately estimated from training data. Then, the probability that the u^{th} leaf node is visited by any particular image is

$$p_{\text{image}}(u) = 1 - (1 - p_{\text{feat}}(u))^{mN_{\text{feat}}} \qquad u = 0, \cdots, k^d - 1 \tag{3.15}$$

assuming the feature descriptors are quantized through the vocabulary tree independently of each other. Then, the length M_u of the u^{th} inverted list is distributed as a binomial random variable:

$$\begin{aligned} p_{M_u}(n) &= \binom{N_{db}}{n} p_{\text{image}}(u)^n \left(1 - p_{\text{image}}(u)\right)^{N_{db}-n} \\ n &= 0, \cdots, N_{db} \qquad\qquad u = 0, \cdots, k^d - 1 \end{aligned} \tag{3.16}$$

assuming the database images are independent of each other. The mean inverted list length is $E[M_u] = \frac{1}{k^d} \sum_{u=0}^{k^d-1} N_{db}\ p_{\text{image}}(u)$. Since $p_{\text{image}}(u)$ decreases as the number of leaf nodes k^d increases, the mean inverted list length decreases as the tree size increases. Next, the mean ordering gain can be calculated to be

$$E[G(M_u)] = \frac{1}{k^d} \sum_{u=0}^{k^d-1} \sum_{n=1}^{N_{db}} \binom{N_{db}}{n} p_{\text{image}}(u)^n \left(1 - p_{\text{image}}(u)\right)^{N_{db}-n} \frac{1}{n} \log_2(n!). \tag{3.17}$$

The mean ordering gains calculated according to Equation 3.17 are reported in Table 3.4 for the three different vocabulary trees. It can be observed that the ordering gain is close to the saving $H(J(\mathbf{i})) - H(R(\mathbf{i}))$ obtained with runlength encoding.

3.3.2 Image Count Coding

There is a one-to-one mapping from the image counts in the inverted lists to the node counts in the database tree histograms. However, the image counts in the inverted lists are assembled in a different order than in the tree histograms. This re-shuffling of the image counts does not affect the distribution of the counts, so the quantizer developed in Section 3.2.2 can also be effectively used here. Consequently, the statistics of the quantized counts are the same as for tree histograms, including the entropy values listed in Table 3.3. In practice, the re-shuffling of the image counts does cause actual entropy codecs like the arithmetic codec, the carryover codec, and the RBUC codec to generate different bitstreams. However, the difference in the memory usage for the quantized image counts compared to the quantized node counts is very small.

3.4 Coding and Retrieval Results

This section reports the large-scale coding and retrieval results for THC and IIC on the MPEG CDVS Dataset described in Appendix A. The vocabulary trees, probability tables for entropy codecs, and parameters for the quantizer of the counts are all generated from an independent training dataset that contains 1M images, as described in Appendix C. Each vocabulary tree is generated by a divisive hierarchical k-means clustering procedure [91] applied on a large set of feature descriptors extracted from the training images.

As we described first in Section 3.1.2, a moderate amount of soft binning significantly increases the probability that matching feature descriptors reach the same leaf node in a large vocabulary tree. Figure 3.14 shows how this effect translates into a noticeable improvement in the retrieval accuracy, for three different vocabulary trees of branch factor $k = 10$ and varying depth $d = 4, 5, 6$. Soft binning ($m = 3$) has $7 - 8$ percent higher accuracy for the MPEG CDVS Dataset in terms of mean precision at rank 1 (PA1) and mean average precision (MAP) compared to hard binning ($m = 1$). Both PA1 and MAP are defined formally in Appendix A. In all subsequent experiments, we will use soft binning for vocabulary tree-based image retrieval.

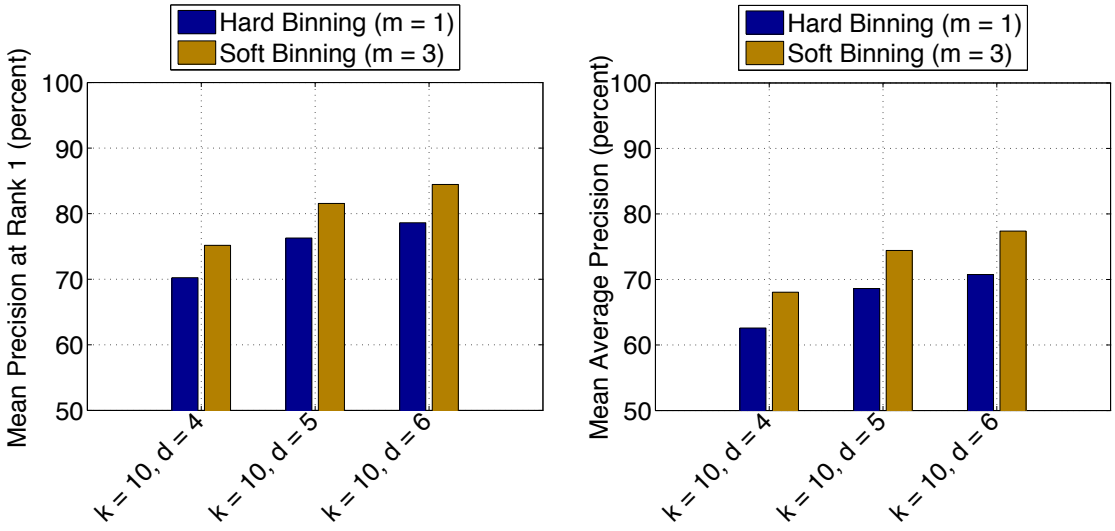


Figure 3.14: (a) Mean precision at rank 1 (PA1) and (b) Mean average precision (MAP) for tree histograms generated with hard binning ($m = 1$) and soft binning ($m = 3$), using the MPEG CDVS Dataset.

Figure 3.14 also shows that the PA1 and MAP values increase as the tree size increases. As we will show later in this section, the query latency also decreases as the tree size increases when an inverted index is used for score computations. For these two reasons, a large vocabulary tree is strongly preferred for the high-accuracy, low-latency MVS applications that we are creating in our work.

Next, we examine the effect of compressing the tree histograms on the retrieval accuracy. Since we losslessly encode the node identifiers in THC and the image identifiers in IIC, there is no impact on the retrieval accuracy from identifier compression. The quantization of the node counts and image counts introduces some distortion, which does slightly change the retrieval accuracy. Figure 3.15 plots the PA1 and MAP for tree histograms with uncompressed counts and compressed counts. It can be observed that retrieval accuracy actually improves slightly, by around 1 percent, after count quantization, because the quantization makes the query and database soft-binned counts more similar for matching images. A similar effect has been independently observed when quantizing the counts with a hard binning scheme [103].

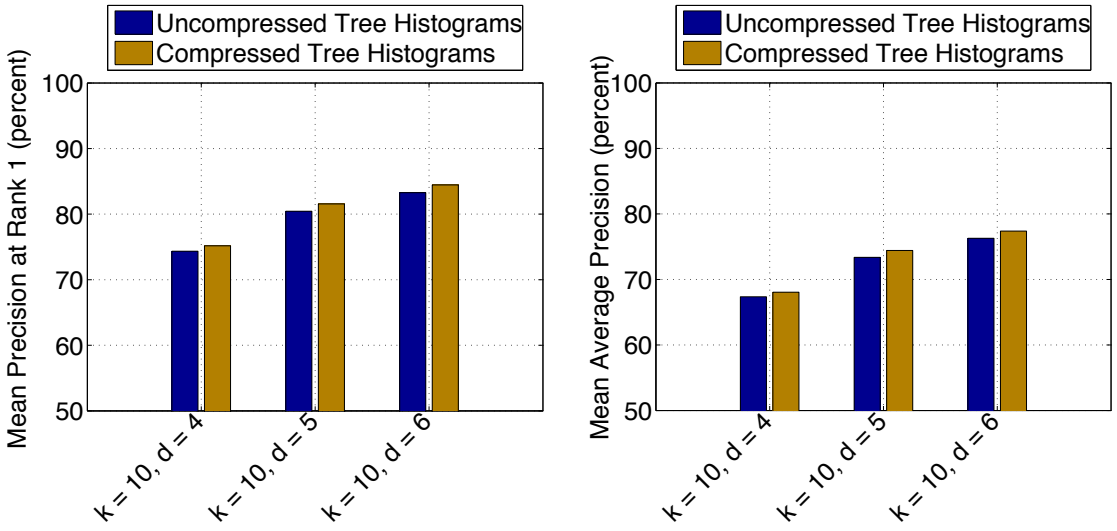


Figure 3.15: (a) Mean precision at rank 1 (PA1) and (b) Mean average precision (MAP) for uncompressed and compressed tree histograms, using the MPEG CDVS Dataset .

The large memory usage of the tree histograms can be substantially reduced using THC. Figure 3.16(a) shows the memory usage of several representations for the database tree histograms in the MPEG CDVS Dataset: (i) an uncompressed database where the node identifiers are stored as integer values and the node counts are stored as floating point values, (ii) a database compressed with THC using the carryover codec described in Section 3.2.3, (iii) a database compressed with THC using the RBUC codec described in Section 3.2.3, and (iv) a database compressed with THC using the arithmetic codec [183]. For the vocabulary tree with $k = 10$ and $d = 6$, THC reduces the memory usage by 4.02×, 4.39×, and 4.91× with the carryover, RBUC, and arithmetic codecs, respectively. Similar compression ratios are obtained for the other two vocabulary trees. Alongside these actual database representations, we also plot the lower bound as determined by the entropies of the node runlengths and quantized node counts. Note that the word-aligned carryover and RBUC codecs, which have fast decoding capabilities, provide savings close to the arithmetic codec and close to the entropy.

Figure 3.16(b) shows an analogous set of results for compressing the inverted

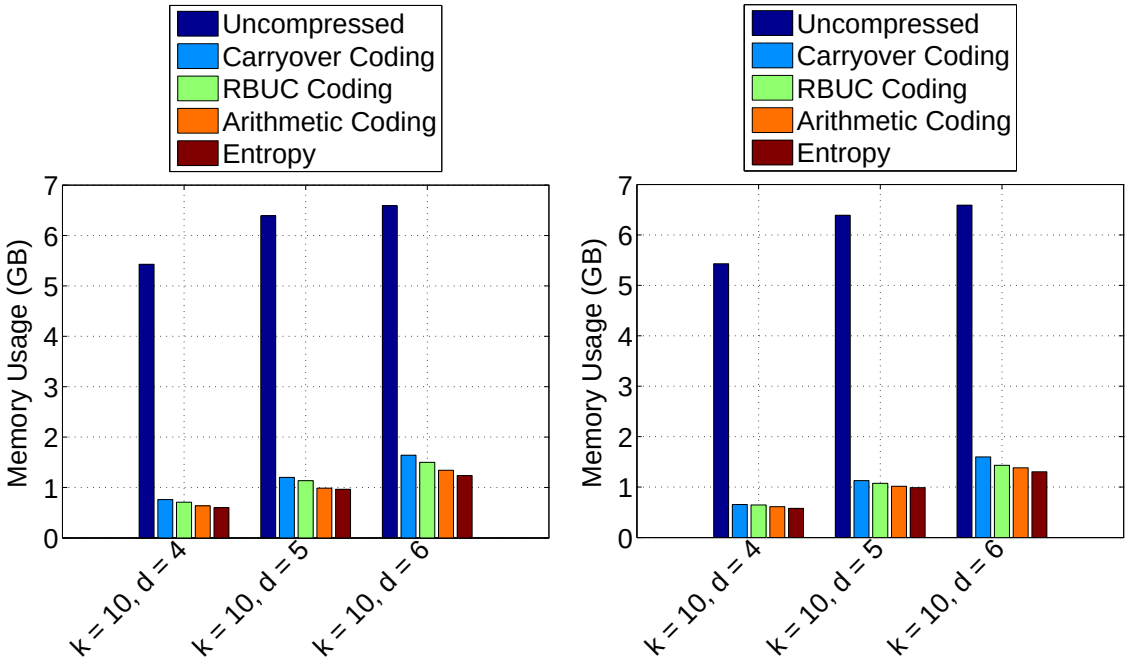


Figure 3.16: Memory usage for (a) tree histogram coding and (b) inverted index coding, with different entropy codecs, using the MPEG CDVS Dataset.

index with IIC. For the tree with $k = 10$ and $d = 6$, the carryover, RBUC, and arithmetic codecs obtain compression ratios of 4.11×, 4.60×, and 4.77×, respectively. Similar compression ratios are obtained for the other two trees. The IIC-compressed inverted index requires about the same amount of memory as the THC-compressed tree histograms. Since the inverted index dramatically reduces the query latency, a low-latency MVS system should use an inverted index for fast score computations.

The benefit of using the word-aligned carryover and RBUC codecs can be seen in Figure 3.17, which plots the decoding latency per query image for THC and IIC. The decoding latency is the time required to decode all compressed identifiers and counts for an image, and the latencies shown in Figure 3.17 are averaged over the 8,313 query images of the MPEG CDVS Dataset. By respecting word boundary constraints and having efficient decoder designs, the carryover and RBUC codecs reduce the decoding time by $4 - 7\times$ compared to the arithmetic codec. For the small loss in compression ratio incurred by the word-aligned codecs, this large reduction in decoding time makes

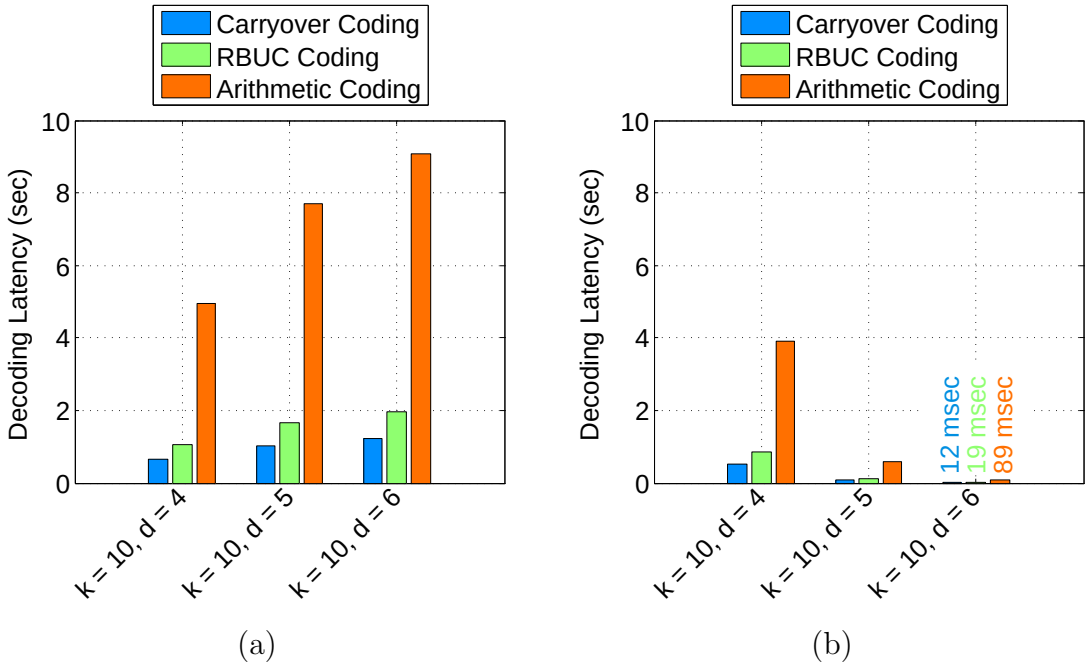


Figure 3.17: Decoding latency for (a) tree histogram coding and (b) inverted index coding, with different entropy codecs, using the MPEG CDVS Dataset. Latencies are measured on a single Intel Xeon 2.4 GHz processor.

the word-aligned codecs attractive for incorporation into low-latency MVS systems.

Figure 3.17 also reveals that the decoding latency increases as the tree size increases when scoring directly with tree histograms using the algorithm in Section 3.1.3. This is because as the tree size increases, the number of node identifiers and node counts corresponding to the positive-count leaf nodes increases, yielding a larger number of symbols per image that need to be decoded. The opposite trend is true when scoring with the inverted index: as the tree size increases, the decoding latency decreases. As shown by the analysis in Section 3.3.1, the mean inverted list length decreases as the tree size increases, causing the number of encoded symbols per list to decrease. At the same time, a query image visits roughly the same number of leaf nodes irrespective of the tree size, and only the inverted lists for the leaf nodes visited by the query image need to be decoded for IIC. Hence, IIC enables a tremendous reduction in the decoding latency by switching to a larger vocabulary tree. This provides a great benefit for fast searching of a compressed image database.

3.5 Summary

In this chapter, we have developed two methods for compressing a large image database consisting of feature histograms: tree histogram coding (THC) and inverted index coding (IIC). These two methods can reduce the database memory usage by a factor of $4-5\times$, without any adverse effect on the image retrieval accuracy. The compressed database actually has 1 percent higher retrieval accuracy than the uncompressed database, because quantization of the feature counts at visual words makes the query and database counts of a matching image pair more similar. Our methods are generally applicable to the many retrieval methods in the computer vision literature that rely on feature histograms. Substantial memory savings are achieved in two stages: (i) efficient compression of the node identifiers or image identifiers by runlength coding, followed by entropy coding of the runlength symbols, and (ii) efficient compression of the node counts or image counts by an entropy-constrained scalar quantizer that is designed for the statistics of soft-binned feature counts, followed by entropy coding of the quantization indices. Our analysis further revealed that the savings by switching from identifiers to runlengths can be interpreted as the amount of information contained in the ordering amongst a set of symbols. Our histogram coding methods can use two different word-aligned codecs that have fast decoding capabilities, which is important for low-latency queries in MVS applications. When an inverted index is employed, we can simultaneously achieve the two goals of increasing the retrieval accuracy and decreasing the query latency simply by increasing the vocabulary tree size.

The large memory savings achieved by our feature histogram coding methods have multiple practical benefits. First, we can deploy a histogram-based MVS system for a large database search on many mobile devices which have only small memory capacities. Second, the memory saved by our coding methods enables the loading of a larger vocabulary tree into RAM, which will improve the retrieval accuracy and reduce the search latency of the entire MVS system. Third, when the database has only a small memory footprint, it is efficient to transfer the database between a mobile device's RAM and SD card, which is important if we want to quickly switch between

different databases for various MVS applications.

The memory usage plotted in Figure 3.16 does not include the memory needed to store the vocabulary tree itself. This is an important issue that we will address further in Chapter 4, where we will show how to create efficient residual-based retrieval methods using much smaller visual codebooks. The residual-based methods can obtain about the same retrieval accuracy as the best performing histogram-based methods described in this chapter.

Chapter 4

Memory-Efficient Feature Residuals

Chapter 3 developed methods for compressing feature histograms and achieved substantial memory savings for large image databases. To achieve a high retrieval accuracy, a large vocabulary tree is needed. The memory usage of the vocabulary tree, which is separate from the memory usage of the feature histograms, becomes an important factor in some common scenarios for MVS applications, such as: (i) storing multiple large vocabulary trees constructed for different applications or different features on the same server, or (ii) storing a large vocabulary tree on a mobile device with a small memory capacity as part of an on-device image retrieval system. Hence, we would like to remove the need for a large visual codebook, while achieving the same retrieval accuracy as methods that require the large codebook. At the same time, we want to achieve greater memory savings than are possible with compressed feature histograms, and we want to further reduce the query latency by performing image matching directly in the compressed domain.

In this chapter, we develop a new class of methods that use feature residuals instead of feature histograms.[1] The main difference between feature histograms and feature residuals is illustrated conceptually in Figure 4.1. Both classes of methods improve in retrieval accuracy as the codebook size increases. We have already seen

[1]Preliminary results of our work have appeared in [37, 40, 36].

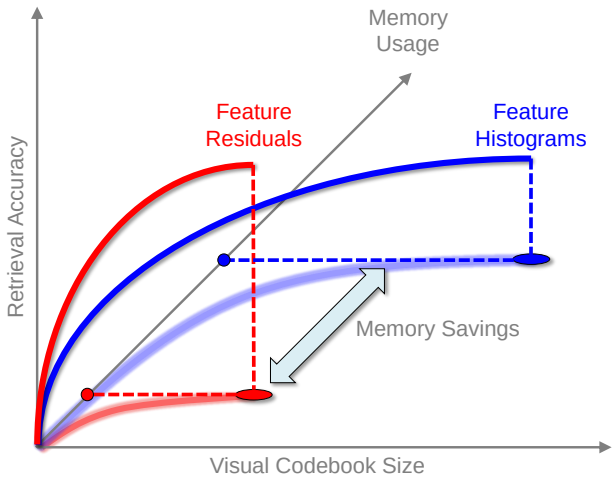


Figure 4.1: Illustration of how feature residuals reduce memory usage compared to feature histograms. Feature residuals can achieve the same retrieval accuracy as feature histograms for a much smaller visual codebook size.

this trend with feature histograms in Chapter 3 and we will show a similar trend for feature residuals in this chapter. Feature residuals encode both where in a Voronoi cell the feature descriptors land and how often the feature descriptors land in that cell, whereas feature histograms just encode the latter information. Due to this key difference, feature residuals can achieve the same retrieval accuracy as feature histograms, while operating with a much smaller codebook and memory footprint.

Our main contribution in this chapter is the development of a residual-based retrieval method that uses a small amount of memory, yields highly accurate retrieval results, and facilitates low-latency searches through a large image database with compressed domain matching. The compact feature residual that we create is called the residual enhanced visual vector (REVV). Figure 4.2 shows an overview of how to first generate a REVV signature for a query image and then compare the query REVV signature against pre-computed database REVV signatures. Each of the operational steps will be covered in detail in the following sections. The version of REVV presented in this chapter contains several new enhancements and optimizations, and therefore this new version obtains the best retrieval performance compared to all previous versions of REVV [37, 40, 36]. Section 4.1 describes the process of generating

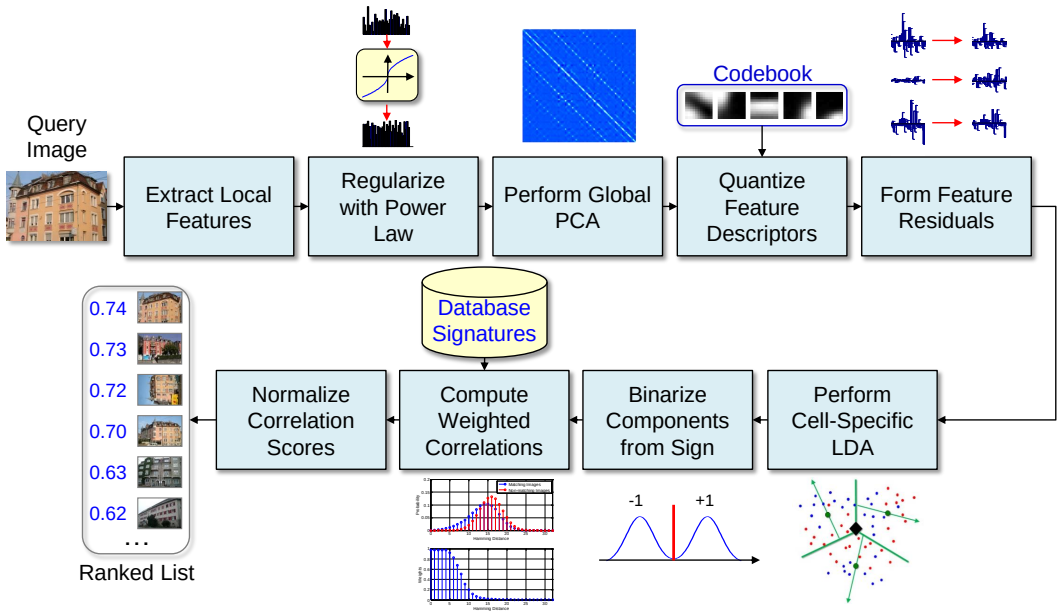


Figure 4.2: Overview of how residual enhanced visual vector (REVV) signatures are generated and compared.

compact and discriminative feature residuals. Then, Section 4.2 explains the process of efficiently comparing residuals in a large database. To better understand how retrieval accuracy varies with several key parameters of REVV and to optimize these parameters in the MVS system, Section 4.3 performs a mathematical analysis of the retrieval performance of REVV signatures. Finally, Section 4.4 reports the memory savings and retrieval results achieved with optimized REVV signatures, including comparisons against the compressed feature histograms from Chapter 3 and against a state-of-the-art global image signature. For the reader's reference, all mathematical symbols used in this chapter are listed in Table 4.1.

4.1 Residual Generation

For each image, we extract N_{feat} local features using the same feature selection method [76] that we used in Chapter 3. First, we apply a power law transformation to these descriptors to reduce the detrimental influence of peaky components, as described

Symbol	Meaning
l	Dimensionality of a feature descriptor
$\mathbf{v}_\alpha$	Power law transformation of vector $\mathbf{v}$ with exponent α
N_{train}	# training feature descriptors
$\mathbf{v}_{\alpha,\text{mean}}$	Global mean vector, using power law with exponent α
$\mathbf{C}_\alpha$	Global covariance matrix, using power law with exponent α
$\boldsymbol{\eta}_i$	Eigenvector of global covariance matrix with i^{th} largest eigenvalue
$\mathbf{T}_{\text{gpca}}$	Global principal component analysis (PCA) matrix
l_{gpca}	Dimensionality after global PCA
$g(\mathbf{v}\|\boldsymbol{\mu}, \boldsymbol{\Sigma})$	PDF of a Gaussian with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
k	# Gaussians in a Gaussian mixture model (GMM)
w_i, $\boldsymbol{\mu}_i$, $\boldsymbol{\Sigma}_i$	Weight, mean, covariance of i^{th} Gaussian in GMM
$\gamma_{i,j}$	Soft assignment of j^{th} descriptor to i^{th} codeword
γ_i	An image's total soft assignment to i^{th} codeword
p_{visit}	Probability that a codeword is visited by an image
$\mathbf{f}_i, \overline{\mathbf{f}_i}$	An image's unnormalized and normalized Fisher vector for the i^{th} codeword
J	Set of indices for training images
$\mathbf{S}_i$	Scatter matrix for the i^{th} codeword
J_m, J_{nm}	Set of indices for matching/non-matching training image pairs
$\mathbf{S}_{nm,i}$	Scatter matrix for matching residuals at the i^{th} codeword
$\mathbf{S}_{nm,i}$	Scatter matrix for non-matching residuals at the i^{th} codeword
$\mathbf{T}_{\text{pca},i}$	Cell-specific principal component analysis (PCA) matrix for i^{th} codeword
$\mathbf{T}_{\text{lda},i}$	Cell-specific linear discriminant analysis (LDA) matrix for i^{th} codeword
l_{pca}, l_{lda}	Dimensionality after cell-specific PCA or LDA
$\mathbf{b}_i$	An image's binarized residual vector for the i^{th} codeword
$N^*_{\text{feat}}(k)$	Optimal # of local features per image for a codebook of size k
$P^*(k)$	Maximal retrieval precision for a codebook of size k
h_i, c_i	Hamming distance and correlation between binary vectors at i^{th} codeword
$w(h)$	Weight for a Hamming distance value of h
I_q, I_{db}	Sets of codewords visited by a query image and a database image
β, γ	Exponents in power law for normalization and correlation weighting
$P_m(h)$, $P_{nm}(h)$	Probability that Hamming distance is h for matching/non-matching pair
Δ_{r1}	Subsampling factor for first round of scoring
N_{r2}	# database images visited in second round of scoring
C_m, C_{nm}	Total correlation for matching/non-matching pair
H_m, H_{nm}	Total Hamming distance for matching/non-matching pair
$p_{\text{visit},m}$, $p_{\text{visit},nm}$	Probability that matching/non-matching pair visit same codeword
$p_{\text{sign},m}$, $p_{\text{sign},nm}$	Probability that matching/non-matching pair have same sign
$N_{\text{visit},m}$, $N_{\text{visit},nm}$	# codewords visited in common by matching/non-matching pair
π_m	Mixing weight in generalized binomial distribution (GDB)
$C^{\max}_m$, $C^{\max}_{nm}$	Max correlation for matching/non-matching database images

Table 4.1: Mathematical symbols used in Chapter 4.

in Section 4.1.1. Next, we apply a global linear transform to reduce the descriptor dimensionality, as described in Section 4.1.2. Then, we quantize the feature descriptors, form quantization residuals, and aggregate the residuals for each codeword, as described in Section 4.1.4. Since the residuals will be binarized based on thresholding, we rotate the residuals by a cell-specific linear transform before binarization to optimally separate matching and non-matching samples, as described in Section 4.1.5. Finally, to finish the generation of the REVV signature, a signed binarization is applied to the residual components to generate a compact binary vector that can be stored efficiently in memory and compared directly in the compressed domain, as described in Section 4.1.6.

4.1.1 Nonlinear Mapping with Power Law

Gradient-based feature descriptors such as SIFT or SURF frequently have peaky components, corresponding to large gradient values in some spatial cells. These peaky values can significantly increase the distance between matching feature descriptors. Applying a properly designed nonlinear mapping in the form of a power law to the descriptor values can reduce the detrimental influence of these peaky components. Statistically, the power law can be interpreted as a variance stabilizing transformation [16]. For a descriptor $\mathbf{v} = [v_1, \cdots, v_l]$, the descriptor after applying the power law transformation is

$$\mathbf{v}_\alpha = [\text{sign}(v_1)\,|v_1|^\alpha, \cdots, \text{sign}(v_l)\,|v_l|^\alpha]\,. \tag{4.1}$$

The power law transformation compands the absolute value of each component while preserving the sign of the component. Figure 4.3 plots the power law transformation for several different values of α. All of the transformations are monotonically nondecreasing functions. As α approaches 1, the power law transformation becomes an identity function. As α approaches 0, the power law transformation makes all components of the descriptor roughly equal to each other in absolute value. Between these two extremes, the power law transformation preserves the relative order of the descriptor components, but reduces the ratio of the maximum component value to the mean component value.

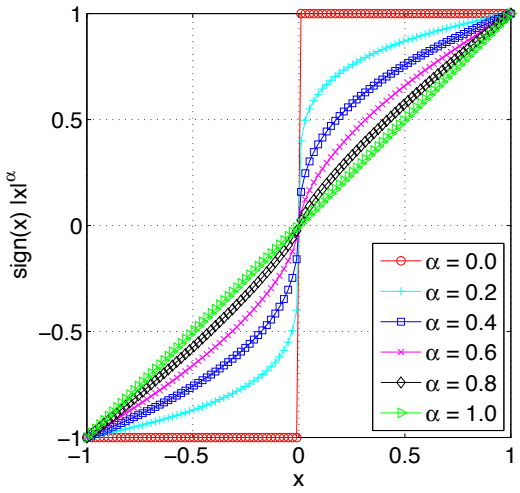


Figure 4.3: Power law transformations for different values of the exponent α.

4.1.2 Global Linear Transform

When learning a codebook from high-dimensional training samples with standard algorithms such as k-means or expectation maximization (EM), the number of training samples required to accurately estimate the codebook parameters increases as the dimensionality increases. Hence, we want to reduce the dimensionality of the feature descriptors, so that the codebook can be effectively estimated from a moderate-sized set of training samples. During dimensionality reduction, we also want to optimally concentrate the energy and salient information of the descriptor into a smaller number of dimensions. These goals can be effectively fulfilled by a global linear transform of the feature descriptor via principal component analysis (PCA). We use the term *global* here to refer to the fact that this transform is applied before the quantization stage and thus no cell-specific statistics are available to design the transform, in contrast to the cell-specific linear transforms we will present later in Section 4.1.5.

To estimate the PCA parameters, we use the independent training dataset described in Appendix C. Suppose we have extracted a set of N_{train} training feature descriptors $\mathbf{v}_1, \cdots, \mathbf{v}_{N_{\text{train}}}$. We estimate the global mean descriptor in the power-law domain as

$$\mathbf{v}_{\alpha,\text{mean}} = \frac{1}{N_{\text{train}}} \sum_{j=1}^{N_{\text{train}}} \mathbf{v}_{\alpha,j} \tag{4.2}$$

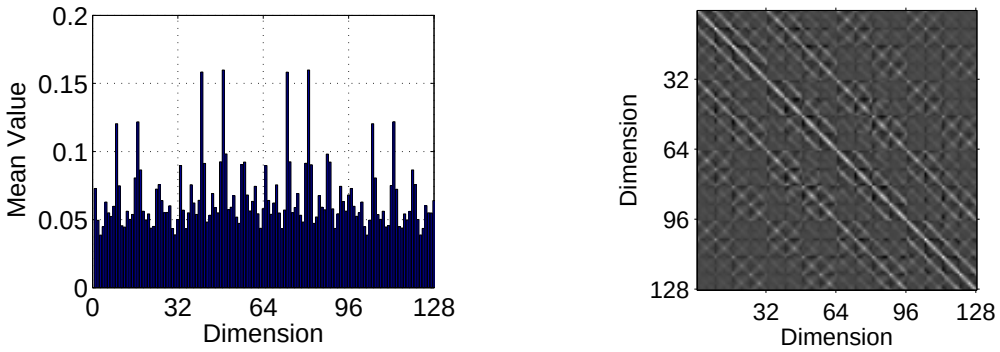


Figure 4.4: (a) Mean vector and (b) Covariance matrix of the 128-dimensional SIFT descriptor, with $\alpha = 0.5$ in the power law transformation.

where $\mathbf{v}_{\alpha,j}$ is the power law-transformed version of descriptor $\mathbf{v}_j$. For example, Figure 4.4(a) shows the mean descriptor for SIFT, with $\alpha = 0.5$ for the power law transformation. Since PCA is most effectively performed on zero-mean samples, we first subtract the mean descriptor $\mathbf{v}_{\alpha,\text{mean}}$ from each power law-transformed training descriptor $\mathbf{v}_{\alpha,j}$. Then, we estimate the global covariance matrix as:

$$\mathbf{C}_\alpha = \frac{1}{N_{\text{train}}} \sum_{j=1}^{N_{\text{train}}} \left(\mathbf{v}_{\alpha,j} - \mathbf{v}_{\alpha,\text{mean}}\right)\left(\mathbf{v}_{\alpha,j} - \mathbf{v}_{\alpha,\text{mean}}\right)^T \tag{4.3}$$

where $\mathbf{v}_{\alpha,j}$ and $\mathbf{v}_{\alpha,\text{mean}}$ are regarded as l-dimensional column vectors, making $\mathbf{C}_\alpha$ an $l \times l$ covariance matrix. We ensure that $N_{\text{train}} \gg l$, so that $\mathbf{C}_\alpha$ is a full-rank matrix. Figure 4.4(b) shows the covariance matrix for SIFT descriptors, again with $\alpha = 0.5$ for the power law transformation. The 8×8 block structures, which appear in the covariance matrix, are due to the fact that the SIFT descriptor is composed of 8-bin orientation histograms for 16 different spatial cells. We compute the first l_{gpca} eigenvectors $\boldsymbol{\eta}_1, \boldsymbol{\eta}_2 \cdots, \boldsymbol{\eta}_{l_{\text{gpca}}} \in \mathbb{R}^l$ of $\mathbf{C}_\alpha$ corresponding to the l_{gpca} largest eigenvalues and define the global PCA matrix as

$$\mathbf{T}_{\text{gpca}} = \left[\boldsymbol{\eta}_1\ \boldsymbol{\eta}_2\ \cdots\ \boldsymbol{\eta}_{l_{\text{gpca}}}\right]^T. \tag{4.4}$$

$\mathbf{T}_{\text{gpca}}$ is an $l_{\text{gpca}} \times l$ matrix that maps each l-dimensional input descriptor to an l_{gpca}-dimensional output coefficient vector.

4.1.3 Codebook Generation

With the global mean vector and PCA matrix estimated in Section 4.1.2, we can generate a set of PCA training vectors

$$\mathbf{v}_{\text{gpca},j} = \mathbf{T}_{\text{gpca}} \left(\mathbf{v}_{\alpha,j} - \mathbf{v}_{\alpha,\text{mean}} \right) \qquad j = 1, \cdots, N_{\text{train}} \tag{4.5}$$

From this set of training vectors, we estimate a Gaussian mixture model (GMM) with k codewords, which has the following probability density function (PDF):

$$p\left(\mathbf{v}\right) = \sum_{i=1}^{k} w_i \, g\left(\mathbf{v} | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right). \tag{4.6}$$

Here, w_i is a scalar, $\boldsymbol{\mu}_i \in \mathbb{R}^{l_{\text{gpca}}}$, and $\boldsymbol{\Sigma}_i \in \mathbb{R}^{l_{\text{gpca}} \times l_{\text{gpca}}}$ are the weight, mean vector, and covariance matrix, respectively, of the i^{th} Gaussian in the GMM. The i^{th} Gaussian has the PDF

$$g\left(\mathbf{v} | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right) = \frac{1}{\left(2\pi\right)^{l_{\text{gpca}}/2} \left|\boldsymbol{\Sigma}_i\right|^{1/2}} \exp\left\{ -\frac{1}{2} \left(\mathbf{v} - \boldsymbol{\mu}_i\right)^T \boldsymbol{\Sigma}_i^{-1} \left(\mathbf{v} - \boldsymbol{\mu}_i\right) \right\}. \tag{4.7}$$

With the parametrization in terms of w_i, $\boldsymbol{\mu}_i$, and $\boldsymbol{\Sigma}_i$ for $i = 1, \cdots, k$, the GMM has a great flexibility in adapting to the complex distributions that naturally occur in the image feature descriptor space.

Given the set of training vectors $\mathbf{v}_{\text{gpca},1}, \cdots, \mathbf{v}_{\text{gpca},N_{\text{train}}}$, the GMM parameters can be iteratively estimated using EM as shown in Algorithm 4.1. In the E-step, the probability of each training vector belonging to each Gaussian is updated. In the M-step, the Gaussian weights, mean vectors, and covariance matrices are updated. We set a maximum number of iterations N_{iter}, but we also allow the option of terminating the algorithm earlier when the change in the log-likelihood $L^{(t)} - L^{(t-1)}$ between two iterations is less than a small threshold ϵ.

At the beginning of EM, a set of initial estimates $w_i^{(0)}, \boldsymbol{\mu}_i^{(0)}, \boldsymbol{\Sigma}_i^{(0)}$ for $i = 1, \cdots, k$

Algorithm 4.1 Expectation maximization (EM) for estimating the Gaussian mixture model (GMM) parameters from a set of training samples.

Require: Training vectors $\mathbf{v}_{\text{gpca},1}, \cdots, \mathbf{v}_{\text{gpca},N_{\text{train}}}$
Require: Initial estimates $w_i^{(0)}, \boldsymbol{\mu}_i^{(0)}, \boldsymbol{\Sigma}_i^{(0)}$ for $i = 1, \cdots, k$

$L^{(0)} = \sum_{j=1}^{N_{\text{train}}} \ln \sum_{i=1}^{k} w_i^{(0)} g\left(\mathbf{v}_{\text{gpca},j} \,|\, \boldsymbol{\mu}_i^{(0)}, \boldsymbol{\Sigma}_i^{(0)}\right)$
for $t = 1 \rightarrow N_{\text{iter}}$ **do**
  E-step
  for $j = 1 \rightarrow N_{\text{train}}$ **do**
    $p_j^{(t)} = \sum_{i=1}^{k} w_i^{(t-1)}\, g\left(\mathbf{v}_{\text{gpca},j} \,|\, \boldsymbol{\mu}_i^{(t-1)}, \boldsymbol{\Sigma}_i^{(t-1)}\right)$
    for $i = 1 \rightarrow k$ **do**
      $\gamma_{i,j}^{(t)} = \frac{w_i^{(t-1)}}{p_j^{(t)}}\, g\left(\mathbf{v}_{\text{gpca},j} \,|\, \boldsymbol{\mu}_i^{(t-1)}, \boldsymbol{\Sigma}_i^{(t-1)}\right)$
    end for
  end for
  M-step
  for $i = 1 \rightarrow k$ **do**
    $q_i^{(t)} = \sum_{j=1}^{N_{\text{train}}} \gamma_{i,j}^{(t)}$
    $w_i^{(t)} = \frac{q_i^{(t)}}{N_{\text{train}}}$
    $\boldsymbol{\mu}_i^{(t)} = \frac{1}{q_i^{(t)}} \sum_{j=1}^{N_{\text{train}}} \gamma_{i,j}^{(t)}\, \mathbf{v}_{\text{gpca},j}$
    $\boldsymbol{\Sigma}_i^{(t)} = \frac{1}{q_i^{(t)}} \sum_{j=1}^{N_{\text{train}}} \gamma_{i,j}^{(t)} \left(\mathbf{v}_{\text{gpca},j} - \boldsymbol{\mu}_i^{(t-1)}\right)\left(\mathbf{v}_{\text{gpca},j} - \boldsymbol{\mu}_i^{(t-1)}\right)^T$
  end for
  $L^{(t)} = \sum_{j=1}^{N_{\text{train}}} \ln \sum_{i=1}^{k} w_i^{(t)} g\left(\mathbf{v}_{\text{gpca},j} \,|\, \boldsymbol{\mu}_i^{(t)}, \boldsymbol{\Sigma}_i^{(t)}\right)$
  if $L^{(t)} - L^{(t-1)} < \epsilon$ **then**
    break
  end if
end for

are required for the weights, mean vectors, and covariance matrices of the k Gaussians. To obtain these initial estimates, we perform k-means clustering on the training vectors. K-means generates a set of cluster assignments $c\left(\mathbf{v}_{\text{gpca},j}\right) \in \{1, \cdots, k\}$ for $j = 1, \cdots, N_{\text{train}}$ for the training vectors , where $q\left(\mathbf{v}_{\text{gpca},j}\right) = i$ means the vector $\mathbf{v}_{\text{gpca},j}$ has been assigned to the i^{th} cluster. From these cluster assignments, we can

compute initial estimates of the GMM parameters as follows:

$$V_i = \{\mathbf{v}_{\text{gpca},j} \ : \ q(\mathbf{v}_{\text{gpca},j}) = i, \quad j = 1, \cdots, N_{\text{train}}\} \tag{4.8}$$

$$w_i^{(0)} = \frac{|V_i|}{N_{\text{train}}} \tag{4.9}$$

$$\boldsymbol{\mu}_i^{(0)} = \frac{1}{|V_i|} \sum_{\mathbf{v} \in V_i} \mathbf{v} \tag{4.10}$$

$$\boldsymbol{\Sigma}_i^{(0)} = \frac{1}{|V_i|} \sum_{\mathbf{v} \in V_i} \left(\mathbf{v} - \boldsymbol{\mu}_i^{(0)}\right) \left(\mathbf{v} - \boldsymbol{\mu}_i^{(0)}\right)^T \tag{4.11}$$

Besides k-means, there are other heuristics for initializing the GMM parameters, e.g., initializing the k mean vectors to be k randomly chosen training vectors, the k weights to be uniform, and the k covariance matrices to be scaled versions of identity matrices. Using k-means to generate the initial parameter estimates, however, effectively reduces the number of EM iterations required to reach convergence and reduces the probability of converging to a local maximum with a small log-likelihood value.

4.1.4 Quantization, Aggregation, and Normalization

Given a set of feature descriptors $\mathbf{v}_1, \cdots, \mathbf{v}_{N_{\text{feat}}}$ for an image, we obtain a set of power law-transformed and global PCA-projected feature vectors $\mathbf{v}_{\text{gpca},1}, \cdots, \mathbf{v}_{\text{gpca},N_{\text{feat}}}$ as described in Sections 4.1.1 and 4.1.2. Now, these feature vectors are quantized to the GMM with parameters w_i, $\boldsymbol{\mu}_i$, and $\boldsymbol{\Sigma}_i$ for $i = 1, \cdots, k$, where the GMM parameters are trained in Section 4.1.3. The quantization performs a soft assignment of the j^{th} feature descriptor in the image to the i^{th} codeword in the GMM as follows:

$$\gamma_{i,j} = \frac{w_i \ g(\mathbf{v}_{\text{gpca},j} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{n=1}^{k} w_n \ g(\mathbf{v}_{\text{gpca},j} \mid \boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n)} \tag{4.12}$$

where the PDF of the i^{th} Gaussian $g(\cdot|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ is defined in Equation 4.7. This soft assignment scheme for the GMM is very similar to the idea of soft binning used for feature histograms in Chapter 3.

The reason why the methods of this chapter are referred to as *feature residual*

methods will now become clearer. To form a discriminative image signature, we use the quantization residuals $\mathbf{v}_{\text{gpca},j} - \boldsymbol{\mu}_i$ between the j^{th} feature descriptor and the i^{th} codeword in a Fisher vector representation. Assuming the N_{feat} feature descriptors are distributed i.i.d. according to the GMM distribution, the log-likelihood of the descriptors can be written as

$$
\begin{aligned}
L\left(\{\mathbf{v}_{\text{gpca},j},\}_{j=1}^{N_{\text{feat}}} \mid \lambda_{\text{gmm}}\right) &= \ln p\left(\{\mathbf{v}_{\text{gpca},j},\}_{j=1}^{N_{\text{feat}}} \mid \lambda_{\text{gmm}}\right) && (4.13)\\
&= \sum_{j=1}^{N_{\text{feat}}} \ln p\left(\mathbf{v}_{\text{gpca},j} | \lambda_{\text{gmm}}\right) && (4.14)\\
&= \sum_{j=1}^{N_{\text{feat}}} \ln \sum_{i=1}^{k} w_i\, g\left(\mathbf{v}_{\text{gpca},j} | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right) && (4.15)
\end{aligned}
$$

where the second line uses the assumption of statistical independence between the features and $\lambda_{\text{gmm}} = \{w_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}_{i=1}^{k}$ represents the GMM parameters. The Fisher vector $\mathbf{f}_i$ for the i^{th} Gaussian in the GMM is defined to be the derivative of the log-likelihood function with respect to the mean vector $\boldsymbol{\mu}_i$:

$$
\begin{aligned}
\mathbf{f}_i &= \frac{\partial L\left(\{\mathbf{v}_{\text{gpca},j},\}_{j=1}^{N_{\text{feat}}} \mid \lambda_{\text{gmm}}\right)}{\partial \boldsymbol{\mu}_i} && (4.16)\\
&= \sum_{j=1}^{N_{\text{feat}}} \frac{\partial}{\partial \boldsymbol{\mu}_i} \ln \sum_{n=1}^{k} w_n\, g\left(\mathbf{v}_{\text{gpca},j} | \boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n\right) && (4.17)\\
&= \sum_{j=1}^{N_{\text{feat}}} \frac{1}{\sum_{n=1}^{k} w_n\, g\left(\mathbf{v}_{\text{gpca},j} | \boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n\right)} \cdot \frac{\partial}{\partial \boldsymbol{\mu}_i} w_i\, g\left(\mathbf{v}_{\text{gpca},j} | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right) && (4.18)\\
&= \sum_{j=1}^{N_{\text{feat}}} \gamma_{i,j} \cdot \frac{\partial}{\partial \boldsymbol{\mu}_i} \left(-\frac{1}{2}\left(\mathbf{v}_{\text{gpca},j} - \boldsymbol{\mu}_i\right)^T \boldsymbol{\Sigma}_i^{-1}\left(\mathbf{v}_{\text{gpca},j} - \boldsymbol{\mu}_i\right)\right) && (4.19)\\
&= \sum_{j=1}^{N_{\text{feat}}} \gamma_{i,j} \cdot \boldsymbol{\Sigma}_i^{-1}\left(\mathbf{v}_{\text{gpca},j} - \boldsymbol{\mu}_i\right). && (4.20)
\end{aligned}
$$

From this derivation, we see that the Fisher vector $\mathbf{f}_i$ for the i^{th} codeword is an l_{gpca}-dimensional vector, which is equivalent to the sum of weighted versions of the residual

vectors $\mathbf{v}_{\text{gpca},j} - \boldsymbol{\mu}_i$. The Fisher vector $\mathbf{f}$ for the entire image is the concatenation of the Fisher vectors for all k codewords in the GMM.

During the residual aggregation process, a measure of the accumulated contributions received by the i^{th} codeword is given by the quantity $\gamma_i = \sum_{j=1}^{N_{\text{feat}}} \gamma_{i,j}$. This quantity is analogous to the total feature count accumulated in a single bin of a feature histogram. When γ_i is low, the i^{th} centroid $\boldsymbol{\mu}_i$ is located far from most of the image's feature descriptors, and the feature residuals $\mathbf{v}_{\text{gpca},j} - \boldsymbol{\mu}_i$ with respect to the i^{th} centroid are not useful signatures for describing the local neighborhood feature statistics. Hence, we discard the Fisher vector for the i^{th} codeword if γ_i is below a small threshold t_γ. Let $p_{\text{visit}} = \sum_{i=1}^{k} w_i \, P\left(\gamma_i > t_\gamma\right)$, a parameter that can be estimated from the training dataset. We will use this parameter p_{visit} later in Section 4.3 when we analyze the retrieval performance of REVV signatures.

We have observed that the norms of the Fisher vectors can vary significantly from one codeword to the next. This variation increases the relative distance between Fisher vectors of matching image pairs, which is undesirable for learning the cell-specific linear transform to be discussed in the next section. Thus, we normalize each codeword's Fisher vector:

$$\bar{\mathbf{f}}_i = \mathbf{f}_i \,/\, \|\mathbf{f}_i\|_2. \tag{4.21}$$

The L_2 normalization for each codeword also reduces the effect of feature bursts generated by repetitive structures in images [103].

In its current form, the Fisher vector is not memory-efficient, because we would require floating point values to store the components of each image's Fisher vector. In the next two sections, we focus on developing a shortened and binarized representation of the Fisher vector that fits compactly in memory.

4.1.5 Cell-Specific Linear Transform

Looking ahead to our final goal of creating a discriminative binary image signature, we will now perform a cell-specific linear transform that accomplishes two purposes: (i) the dimensionality of the Fisher vector is reduced while the most salient information is preserved, and (ii) the projection directions are chosen to better distinguish

between matching and non-matching image pairs. Now that the feature descriptors have been quantized to different codewords, we can exploit cell-specific statistics that describe the local neighborhoods around the GMM centroids. This is in contrast to the global linear transform used in Section 4.1.2, where cell-specific statistics were not yet available and only the global feature statistics could be utilized.

In our first attempt at designing a cell-specific linear transform, we choose the projection direction in each cell according to a cell-specific PCA. To calculate the projection direction, we use a set of images contained in the training dataset described in Appendix C. Let $J = \{j_1, \cdots, j_N\}$ denote a set of N training image indices. The cell-specific PCA solves the following optimization problem:

$$\underset{\mathbf{w}_i}{\text{maximize}} \quad \sum_{j \in J} \left(\mathbf{w}_i^T \, \bar{\mathbf{f}}_{j,i}\right)^2 \tag{4.22}$$

$$\text{s.t.} \quad \|\mathbf{w}_i\|_2 = 1 \tag{4.23}$$

where $\bar{\mathbf{f}}_{j,i}$ is the normalized Fisher vector of the i^{th} Gaussian for the image indexed by j. The solution is given by the eigenvector $\boldsymbol{\eta}_{i,1}$ of the scatter matrix $\mathbf{S}_i$ that has the largest eigenvalue $\lambda_{i,1}$:

$$\mathbf{S}_i = \sum_{j \in J} \bar{\mathbf{f}}_{j,i} \, \bar{\mathbf{f}}_{j,i}^T \tag{4.24}$$

$$\mathbf{S}_i \boldsymbol{\eta}_{i,1} = \lambda_{i,1} \boldsymbol{\eta}_{i,1}. \tag{4.25}$$

If instead we want to project onto $l_{\text{pca}} > 1$ directions, we would select the l_{pca} eigenvectors $\boldsymbol{\eta}_{i,1}, \cdots, \boldsymbol{\eta}_{i,l_{\text{pca}}}$ corresponding to the l_{pca} largest eigenvalues. Then, the transform matrix for cell-specific PCA in the i^{th} cell is

$$\mathbf{T}_{\text{pca},i} = \left[\boldsymbol{\eta}_{i,1} \, \cdots \, \boldsymbol{\eta}_{i,l_{\text{pca}}}\right]. \tag{4.26}$$

Note that $\mathbf{T}_{\text{pca},i} \in \mathbb{R}^{l_{\text{pca}} \times l_{\text{gpca}}}$, so it maps an l_{gpca}-dimensional Fisher vector into an l_{pca}-dimensional coefficient vector.

Cell-specific PCA chooses the direction of maximal variance for projection and minimizes the L_2 approximation error relative to the original Fisher vector. However,

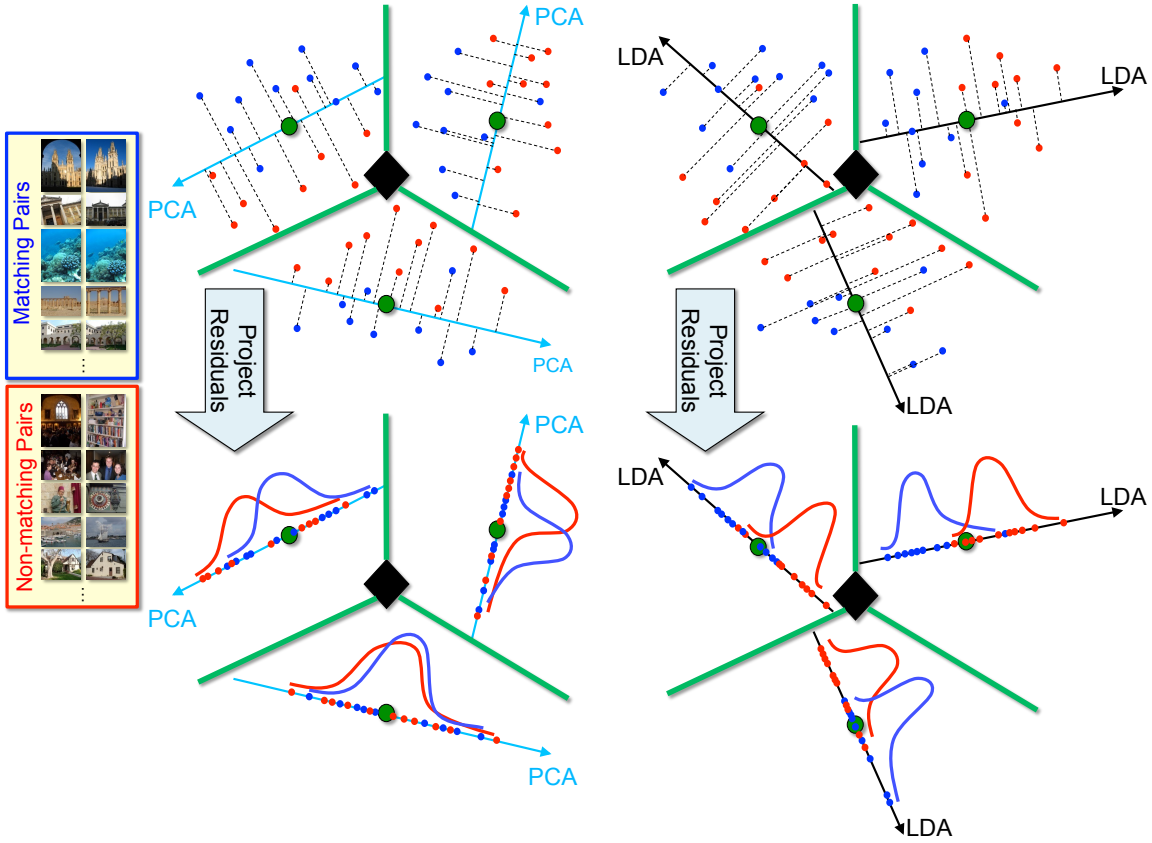


Figure 4.5: Projection of feature residuals using cell-specific principal component analysis (PCA) and cell-specific linear discriminant analysis (LDA).

the left side of Figure 4.5 illustrates a potential drawback of cell-specific PCA for a toy example with $k = 3$ codewords. Because the optimization objective here makes no attempt to separate matching and non-matching samples, the matching and non-matching samples could be closely mixed together after projection for some cells. This can lead to a lower retrieval accuracy if we use the PCA-projected Fisher vectors to compare different images.

A better approach in designing a cell-specific linear transform is to incorporate information about matching and non-matching image pairs directly into the optimization objective. Let the indices of N_m different matching image pairs from the

training dataset be denoted as

$$J_m = \{(j_{m,1,1}, j_{m,1,2}), \cdots, (j_{m,N_m,1}, j_{m,N_m,2})\}. \tag{4.27}$$

Similarly, let the indices of N_{nm} non-matching image pairs from the training dataset be denoted as

$$J_{nm} = \{(j_{nm,1,1}, j_{nm,1,2}), \cdots, (j_{nm,N_{nm},1}, j_{nm,N_{nm},2})\}. \tag{4.28}$$

We will use $N_m = N_{nm}$ during training. Then, a cell-specific linear discriminant analysis (LDA) solves the following optimization problem:

$$\underset{\mathbf{w}_i}{\text{maximize}} \quad \frac{\sum_{(j_1,j_2)\in J_{nm}} \left(\mathbf{w}_i^T \left(\bar{\mathbf{f}}_{j_1,i} - \bar{\mathbf{f}}_{j_2,i}\right)\right)^2}{\sum_{(j_1,j_2)\in J_m} \left(\mathbf{w}_i^T \left(\bar{\mathbf{f}}_{j_1,i} - \bar{\mathbf{f}}_{j_2,i}\right)\right)^2} \tag{4.29}$$

$$\text{s.t.} \quad \|\mathbf{w}_i\|_2 = 1 \tag{4.30}$$

Since we enforce $N_m = N_{nm}$, the ratio in Equation 4.29 can be interpreted as the ratio of (i) the variance of distances between non-matching Fisher vectors to (ii) the variance of distances between matching Fisher vectors. Note that our problem formulation here is different from the formulation of classical Fisher LDA. The solution is given by the generalized eigenvector $\boldsymbol{\eta}_{i,1}$ of two scatter matrices, corresponding to the largest eigenvalue $\lambda_{i,1}$:

$$\mathbf{S}_{m,i} = \sum_{(j_1,j_2)\in J_m} \left(\bar{\mathbf{f}}_{j_1,i} - \bar{\mathbf{f}}_{j_2,i}\right)\left(\bar{\mathbf{f}}_{j_1,i} - \bar{\mathbf{f}}_{j_2,i}\right)^T \tag{4.31}$$

$$\mathbf{S}_{nm,i} = \sum_{(j_1,j_2)\in J_{nm}} \left(\bar{\mathbf{f}}_{j_1,i} - \bar{\mathbf{f}}_{j_2,i}\right)\left(\bar{\mathbf{f}}_{j_1,i} - \bar{\mathbf{f}}_{j_2,i}\right)^T \tag{4.32}$$

$$\mathbf{S}_{nm,i}\boldsymbol{\eta}_{i,1} = \lambda_{i,1}\mathbf{S}_{m,i}\boldsymbol{\eta}_{i,1}. \tag{4.33}$$

Like in the cell-specific PCA case, if we want to project onto $l_{\text{lda}} > 1$ directions, we would select the l_{lda} generalized eigenvectors corresponding to the l_{lda} largest

eigenvalues. The transform matrix for cell-specific LDA in the i^{th} cell is

$$\mathbf{T}_{\text{lda},i} = [\boldsymbol{\eta}_{i,1} \cdots \boldsymbol{\eta}_{i,l_{\text{lda}}}]. \tag{4.34}$$

Note that $\mathbf{T}_{\text{lda},i} \in \mathbb{R}^{l_{\text{lda}} \times l_{\text{gpca}}}$, so it maps an l_{gpca}-dimensional Fisher vector into an l_{lda}-dimensional coefficient vector.

The right side of Figure 4.5 shows cell-specific LDA projection for the same toy example of $k = 3$ codewords. Matching and non-matching Fisher vectors are now much better separated after cell-specific LDA compared to cell-specific PCA. This is especially important for large-scale image retrieval, where for a query Fisher vector, we want to be able to accurately distinguish between the few matching database Fisher vectors and the many non-matching database Fisher vectors. Cell-specific LDA generally removes dimensions of the Fisher vector that contain noise and intra-class variations. The removed dimensions are generally not helpful for separating between matching and non-matching image pairs.

4.1.6 Signed Binarization

Having obtained a shorter and more discriminative image signature using the cell-specific LDA of the last section, we can now binarize the image signature based on the sign of the LDA coefficients. Let $\mathbf{f}_{\text{lda},i} = \mathbf{T}_{\text{lda},i}\,\mathbf{f}_i$ be the coefficient vector that results from the projection of the i^{th} Fisher vector using cell-specific LDA. The signed binarization of this coefficients vector is defined to be

$$\mathbf{b}_i = \left[\text{sign}\left\{(\mathbf{f}_{\text{lda},i})_1\right\}\ \text{sign}\left\{(\mathbf{f}_{\text{lda},i})_2\right\}\ \cdots\ \text{sign}\left\{(\mathbf{f}_{\text{lda},i})_{l_{\text{lda}}}\right\}\right] \tag{4.35}$$

Hence, $\mathbf{b}_i \in \{-1, +1\}^{l_{\text{lda}}}$ is an l_{lda}-dimensional vector containing just -1 and $+1$ elements. As we will show in the results of Section 4.4, despite the extremely coarse quantization applied in the signed binarization of the LDA coefficients, the set of binary vectors $\{\mathbf{b}_i\}_{i=1}^{k}$ remains a highly discriminative image signature capable of accurate search through a large database.

There are two significant benefits gained from the signed binarization. First,

the binary vector $\mathbf{b}_i$ requires just l_{lda} bits to store in memory. For example, when $l_{\text{lda}} = 32$, the entire vector $\mathbf{b}_i$ fits compactly into a 32-bit computer word. Second, binary vectors can be efficiently compared in the compressed domain. The Hamming distance between two binary vectors can be computed very quickly using atomic XOR and POPCNT instructions, as we show later in Section 4.2.

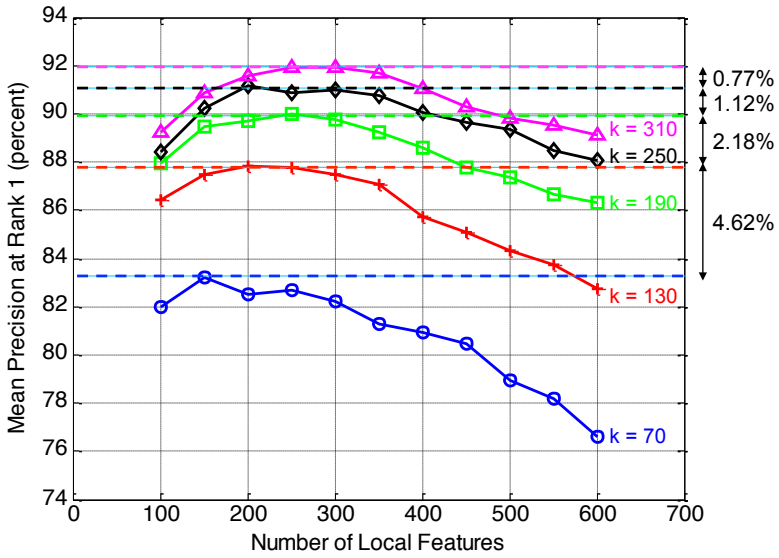
4.1.7 Codebook and Feature Selection

In our discussion so far, we have assumed that the number of codewords k and the number of local features N_{feat} extracted from each image are both set to some reasonable values. Now, we explain how these two parameters can be optimally selected to yield the best retrieval accuracy. We perform an experiment where we systematically vary the number of codewords from $k = 70$ to $k = 310$ in increments of 60 and vary the number of local features from $N_{\text{feat}} = 100$ to $N_{\text{feat}} = 600$ in increments of 50. For each pair of parameters (k, N_{feat}), we measure the retrieval accuracy for all of the query images and 20K database images in the MPEG CDVS Dataset described in Appendix A. Details about how REVV signatures are compared to generate the retrieval accuracy measurements will be described subsequently in Section 4.2, so that we can focus on the effects of changing the number of codewords and number of local features in this section.

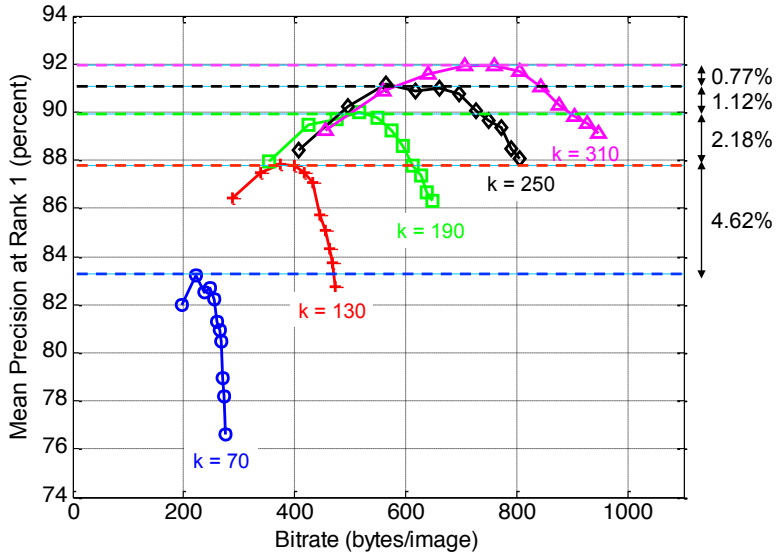
Figure 4.6(a) plots the retrieval accuracy in terms of the mean precision at rank 1 (PA1) for the different codebook sizes versus the number of local features, while Figure 4.6(b) plots the same set of curves except that the horizontal axis is now converted to the bitrate (in bytes/image) required to store each image's REVV signature in memory. Several important performance trends can be observed in these plots:

1. For a fixed codebook size k, there exists an optimal number of local features $N^*_{\text{feat}}(k)$ per image which yields an maximal precision $P^*(k)$. The maximal precisions are indicated by the dashed horizontal lines in Figure 4.6.

2. As the codebook size k increases, the maximal precision $P^*(k)$ increases, but at a diminishing rate. The gaps between the maximal precisions of the different codebooks are shown on the right side of the plots in Figure 4.6. The values



(a)



(b)

Figure 4.6: (a) Mean precision at rank 1 (PA1) versus the number of local features. (b) PA1 versus the bitrate in bytes/image. Precision values are reported for query images and a database of 20K images in the MPEG CDVS Dataset. REVV signatures for five different codebook sizes ($k = 70, 130, 190, 250, 310$) are evaluated. The horizontal dashed lines denote the maximal precision values achieved for each codebook size.

of the gaps are $P^*(130) - P^*(70) = 4.62$ percent, $P^*(190) - P^*(130) = 2.18$ percent, $P^*(250) - P^*(190) = 1.12$ percent, $P^*(310) - P^*(250) = 0.77$ percent. Hence, there is a diminishing benefit gained from increasing the codebook size.

3. As the codebook size k increases, the bitrate required to store each REVV signature in memory increases. Coupled with the second trend, there are diminishing gains in the retrieval precision by steadily increasing the memory usage of the database.

The reasons for why there exists an optimal number of local features for each codebook and why there are diminishing gains in the maximal precision as the codebook size increases will be discussed in Section 4.3, when we perform a statistical analysis of the retrieval performance for REVV signatures. However, we can already draw the following conclusions from the results of Figure 4.6: for achieving the optimal retrieval accuracy, we need to carefully choose the codebook size k to match the desired database memory usage and choose the optimal number of local features $N^*_{\text{feat}}(k)$ for that codebook size.

4.2 Residual Comparison

In Section 4.1, we described the steps for generating a discriminative and compact REVV signature from a set of local feature descriptors. Each REVV signature is a binary vector that stores the signs of the cell-specific LDA coefficients. Now, we show how these binary vectors can be efficiently compared to yield informative estimates about the similarity between two images. First, Section 4.2.1 describes how image correlation scores are computed directly in the compressed domain. Then, Section 4.2.2 develops a weighting scheme that considers the probability of observing different Hamming distances from matching and non-matching image pairs. This weighting scheme gives more emphasis to observations that are more likely to have originated from matching pairs and improves the image matching performance. Finally, to avoid an exhaustive linear search through a large database, Section 4.2.3

creates a multi-round scoring algorithm that can speed up the database search substantially, without harming the retrieval accuracy.

4.2.1 Compressed Domain Matching

Figure 4.7 gives an overview of compressed domain matching for REVV signatures. After an LDA coefficient vector is binarized, the binary vector is packed into the bits of an integer variable in memory. The Hamming distance between two binary vectors for the same codeword can be computed in two simple steps: (i) a logical XOR operation that finds the positions where the bits of the two binary vectors differ, followed by (ii) a POPCNT instruction that counts how many bits differ. Given binary vectors $\mathbf{b}_{1,i}$ and $\mathbf{b}_{2,i}$ for two images at the i^{th} codeword, the Hamming distance h_i is given by

$$\begin{aligned} h_i &= \text{POPCNT}\left(\text{XOR}\left(\mathbf{b}_{1,i}, \mathbf{b}_{2,i}\right)\right) \\ h_i &\in \{0,\, 1,\, \cdots,\, l_{\text{lda}}\} \end{aligned} \tag{4.36}$$

Most computer instruction sets contain XOR and POPCNT as atomic operations, so calculating the Hamming distance is very fast. This Hamming distance can be equivalently converted into a correlation value:

$$\begin{aligned} c_i &= l_{\text{lda}} - 2h_i \\ c_i &\in \{-l_{\text{lda}},\, -l_{\text{lda}} + 2,\, \cdots,\, l_{\text{lda}} - 2,\, l_{\text{lda}}\} \end{aligned} \tag{4.37}$$

Note that because the Hamming distance takes a set of $l_{\text{lda}} + 1$ discrete values, the correlation also takes a set of $l_{\text{lda}} + 1$ discrete values.

Let I_q and I_{db} be the sets of codewords visited by the query and database images, respectively. The overall correlation between the two images is given by

$$C = \frac{1}{\left(|I_q| \cdot |I_{db}|\right)^{\beta}} \sum_{i \in I_q \cap I_{db}} w\left(h_i\right) c_i \tag{4.38}$$

where $w\left(h_i\right)$ is a weight that depends on the Hamming distance h_i that we will describe in the next section. The normalization factor $1/\left(|I_q| \cdot |I_{db}|\right)^{\beta}$ applies a power

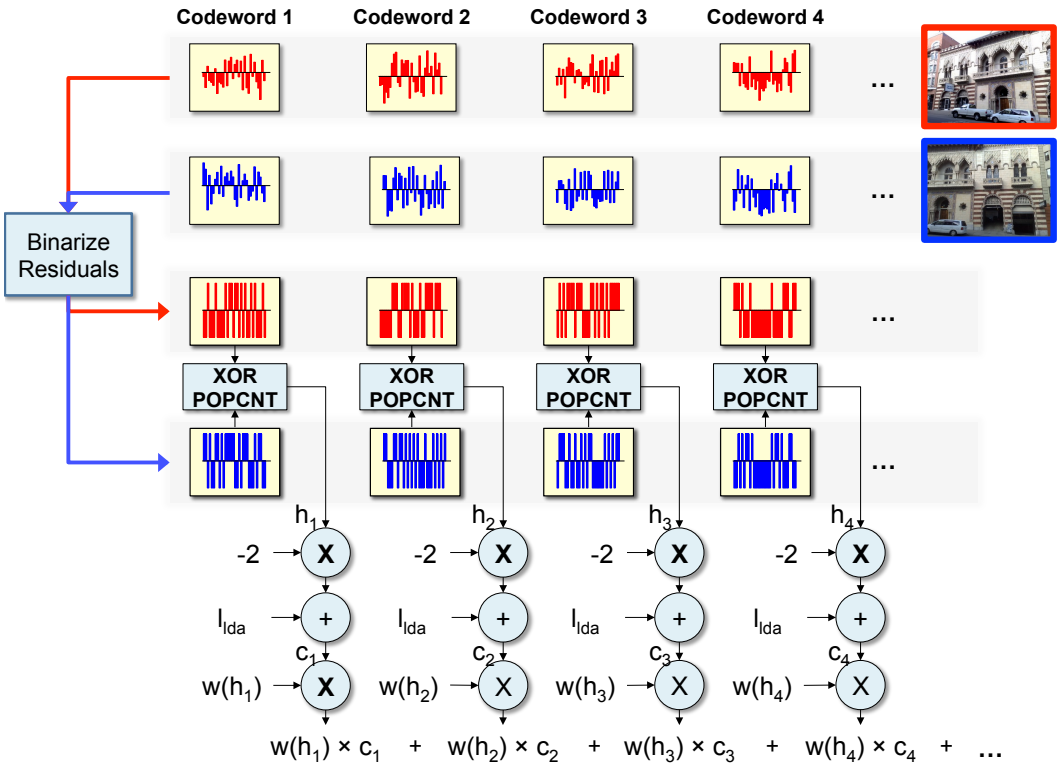


Figure 4.7: Binarization of residuals based on the polarity and computation of correlation scores in the compressed domain. The Hamming distance between binary residual vectors is computed using XOR and POPCNT instructions and then converted to a weighted correlation value.

law transformation with exponent β to the product of the codeword visit counts $|I_q| \cdot |I_{db}|$ to reduce the influence of peaky visit counts, in the same spirit as the power law transformation of Section 4.1.1.

We can obtain further computational savings by recognizing that the weighted correlation $w(h_i)\, c_i$ has only $l_{\text{lda}} + 1$ possible values, which can be easily stored in and subsequently read from a lookup table. Then, evaluation of Equation 4.38 requires only:

- At most k separate XOR, POPCNT, and table lookups to compute the individual weighted correlations.

- At most $k-1$ separate additions to compute the sum of the weighted correlations.
- One multiplication, one exponentiation, and one division to compute the overall score normalization.

4.2.2 Weighted Correlations

Similar to the process of designing a cell-specific LDA in Section 4.1.5, we can take advantage of differing statistics for matching and non-matching image pairs to design a weighting function for the correlation scores. The top of Figure 4.8 plots the probability $P_m(h)$ and $P_{nm}(h)$ of observing a Hamming distance h at any codeword for a matching or a non-matching image pair, respectively. These statistics are generated from images in the training dataset in Appendix C. It is more probable to observe smaller Hamming distance values for matching pairs than for non-matching pairs.

We exploit the important differences between $P_m(h)$ and $P_{nm}(h)$ by creating a weighting function $w(h)$ that decreases as h increases:

$$w(h) = \left(\frac{P_m(h)}{P_m(h) + P_{nm}(h)} \right)^{\gamma} \tag{4.39}$$

where $\gamma > 1$ is a parameter that controls how quickly $w(h)$ decreases as h increases. The bottom of Figure 4.8 shows the shape of the weighting function that results for $\gamma = 2$. In this weighting function, observations with large Hamming distances have weights close to 0 and are thus severely discounted, because these large Hamming distances are likely to arise from non-matching image pairs. Conversely, observations with small Hamming distances have weights close to 1 and are thus strongly rewarded, because these small Hamming distances are likely to arise from matching image pairs. This weighting function is especially beneficial for accurate image retrieval from a large database, where most of the observations encountered are caused by non-matching database images and should be discounted. After the weighting function is applied, the correlation scores assigned to matching database images stand out more prominently in a ranked list.

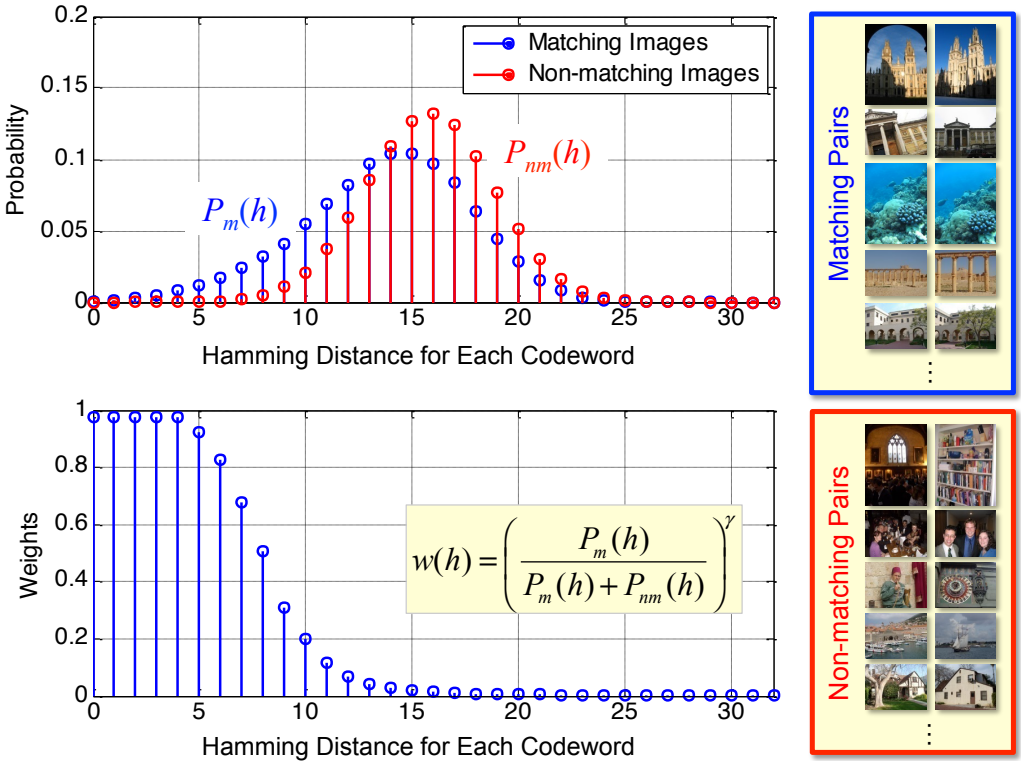


Figure 4.8: (Top) Distribution of Hamming distances between residual vectors at a single codeword for matching and non-matching image pairs. (Bottom) Correlation weights for Hamming distances at a single codeword.

4.2.3 Multi-Round Scoring

The compressed domain matching procedure described in Section 4.2.1 is fast. However, when our goal is to obtain a shortlist of the best matching database images, a brute-force linear search where every database REVV signature is compared against the query REVV signature is both unnecessary and computationally wasteful. To further speed up the search through a large database and avoid an exhaustive evaluation of the REVV correlation scores for most database images, we develop a multi-round scoring algorithm. Figure 4.9 shows an instance of this algorithm with two rounds of scoring being used to compare a query REVV signature to the database REVV signatures.

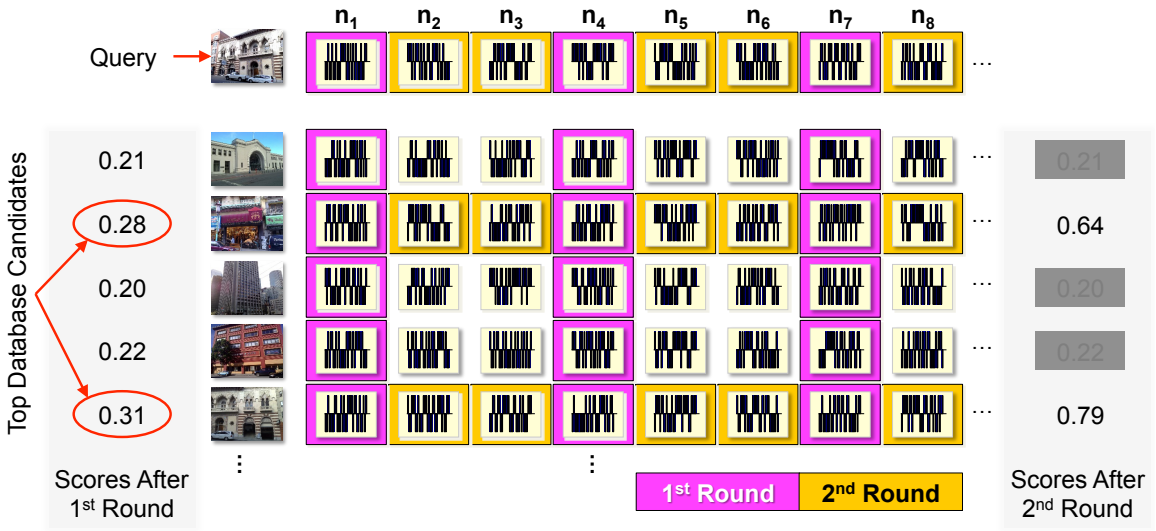


Figure 4.9: Computation of correlation scores in a large database with multi-round scoring. In the first round, partial correlation scores are computed for all database images. Then, a small set of the most promising database candidates are selected for further exploration. In the second round, full correlation scores are computed for this small set of top database candidates.

In the first round of scoring, the correlation scores are computed only for codewords $I_{r1} = \left\{1,\, \Delta_{r1}+1,\, 2\Delta_{r1}+1,\, \cdots,\, \left\lfloor \frac{k-1}{\Delta_{r1}} \right\rfloor \Delta_{r1}+1 \right\}$. Then, only $\lfloor (k-1)/\Delta_{r1} \rfloor + 1$ out of the k codewords are visited. The example in Figure 4.9 uses $\Delta_{r1} = 3$. Based on this partial traversal through the codewords in the first round, we obtain a set of partial correlation scores for all database images:

$$C_{r1}^{j} = \frac{1}{\left(|I_q| \cdot |I_{db}^{j}|\right)^{\beta}} \sum_{i \in I_{r1} \cap I_q \cap I_{db}^{j}} w\left(h_i^j\right) c_i^j \qquad j = 1, \cdots, N_{db} \tag{4.40}$$

where h_i^j and c_i^j are the Hamming distance and correlation, respectively, between the query REVV signature and the j^{th} database REVV signature at the i^{th} codeword. Additionally, I_q and I_{db}^j are the sets of codewords visited in the query REVV signature and j^{th} database REVV signature, respectively.

In the second round of scoring, we sort the partial correlation scores $\left\{C_{r1}^{j}\right\}_{j=1}^{N_{db}}$ obtained from the first round in descending order. After sorting, we identify the N_{r2}

Operation	Two-Round Scoring	Exhaustive Scoring
XOR	$(N_{db} - N_{r2}) \left(\left\lfloor \frac{k-1}{\Delta_{r1}} \right\rfloor + 1 \right) + N_{r2}\, k$	$N_{db}\, k$
POPCNT	$(N_{db} - N_{r2}) \left(\left\lfloor \frac{k-1}{\Delta_{r1}} \right\rfloor + 1 \right) + N_{r2}\, k$	$N_{db}\, k$
Addition	$(N_{db} - N_{r2}) \left\lfloor \frac{k-1}{\Delta_{r1}} \right\rfloor + N_{r2}\, k$	$N_{db}\ (k-1)$
Multiplication	N_{db}	N_{db}
Exponentiation	N_{db}	N_{db}
Division	$N_{db} + N_{r2}$	N_{db}
Sorting	$O\left(N_{db} \log_2 N_{db} + N_{r2} \log_2 N_{r2}\right)$	$O\left(N_{db} \log_2 N_{db}\right)$

Table 4.2: Worst-case computational complexity for a two-round scoring algorithm and an exhaustive scoring algorithm.

images $\{j_{r2,1}, \cdots, j_{r2,N_{r2}}\}$ with the highest partial correlation scores, which are worthy of further exploration in the second round. By setting $N_{r2} \ll N_{db}$, the number of database images considered in the second round is much smaller than the total number of images in the database. We finish computing the full correlation scores for just these N_{r2} images, by visiting the remaining codewords $I_{r2} = \{1, 2, \cdots, k\} \setminus I_{r1}$ that were not visited in the first round:

$$\begin{aligned} C_{r2}^{j} &= C_{r1}^{j} + \frac{1}{\left(|I_q| \cdot |I_{db}^{j}|\right)^{\beta}} \sum_{i \in I_{r2} \cap I_q \cap I_{db}^{j}} w\left(h_i^j\right) c_i^j \\ j &\in \{j_{r2,1}, \cdots, j_{r2,N_{r2}}\} \end{aligned} \tag{4.41}$$

Table 4.2 lists the maximal number of operations required for (i) a two-round scoring algorithm and (ii) an exhaustive scoring algorithm where full correlations are computed for all database images. The normalization factors in Equation 4.41 during the second round have already been computed in Equation 4.40 during the first round, but an extra set of divisions is still required. Although two-round scoring requires more division and sorting operations, two-round scoring can obtain substantial savings in the number of XORs, POPCNTs, and additions. We will show in the results of Section 4.4 that by carefully selecting Δ_{r1} and N_{r2}, we can obtain a substantial speedup in the database search without reducing the retrieval accuracy.

4.3 Analysis of Retrieval Performance

As we showed in Section 4.1.7, the proper selection of the number of codewords k and the number of local features per image N_{feat} can have a significant impact on the retrieval accuracy for REVV signatures. We will now perform a mathematical analysis of the retrieval performance of REVV signatures, first to explain the important characteristics observed in Section 4.1.7 and then to guide the optimal selection of the REVV parameters. An analysis for the retrieval performance of histogram-based retrieval methods has been developed by Chandrasekhar [30]. We are the first to develop and present a detailed analysis for the retrieval performance of residual-based retrieval methods.

First, Section 4.3.1 creates statistical models for the REVV correlation scores of matching and non-matching image pairs. Then, Section 4.3.2 generates a model of the retrieval accuracy in searching a large database, based on the model of the REVV correlations scores. Our analysis accurately predicts how the retrieval accuracy varies with the number of codewords k and the number of local features N_{feat}. Based on this analysis, we can choose both the optimal number the codewords and the optimal number of local features to meet any target memory budget and achieve the overall best system performance.

4.3.1 Modeling Correlation Scores

First, we develop a statistical model for the correlations between binary residual vectors. Our model captures the most important first-order effects in how the correlation scores change as the number of codewords k or the number of local features N_{feat} changes. Some second-order effects caused by score normalization and correlation weighting are ignored in this analysis to keep the number of parameters in the model to a minimum and concentrate on the important trade-offs. When excluding the score normalization and correlation weighting steps, the correlation C_{nm} for a

non-matching image pair and C_m for a matching image pair are given by:

$$\begin{aligned} C_\theta &= \sum_{i=1}^{N_{\text{visit},\theta}} C_{\theta,i} \\ \theta &\in \{nm, m\} = \{\text{non-matching}, \text{matching}\} \end{aligned} \tag{4.42}$$

where $C_{\theta,i}$ is the correlation between binary residual vectors at a single codeword and $N_{\text{visit},\theta}$ is the number of codewords visited in common by the pair of images. Equivalently, if we know the Hamming distance $H_{\theta,i}$ between binary residual vectors at a single codeword, we can compute the codeword-level correlation as $C_{\theta,i} = l_{\text{lda}} - 2H_{\theta,i}$, where l_{lda} is the dimensionality of the residual vector after cell-specific LDA. The image-level correlation can then be rewritten as

$$C_\theta = N_{\text{visit},\theta}\, l_{\text{lda}} - 2H_\theta \qquad \theta \in \{nm, m\} \tag{4.43}$$

$$H_\theta = \sum_{i=1}^{N_{\text{visit},\theta}} H_{\theta,i} \tag{4.44}$$

where H_θ is an image-level Hamming distance.

The number of codewords $N_{\text{visit},\theta}$ visited in common by the pair of images can be modeled as a binomial random variable:

$$p_{N_{\text{visit},\theta}}(n) = \binom{k}{n} p_{\text{visit},\theta}^{n} \left(1 - p_{\text{visit},\theta}\right)^{k-n} \qquad \theta \in \{nm, m\}$$

where $p_{\text{visit},\theta}$ is the probability that a codeword is visited in common by the pair of images. For non-matching images, whether or not one image visits a codeword has no effect on whether or not the other image visits the same codeword. Therefore, we have $p_{\text{visit},nm} = p_{\text{visit}}^2$, where p_{visit} is the probability that a codeword is visited by a single image, as defined in Section 4.1.4. For matching images, if one image visits a codeword, then the other matching image will visit the codeword with a probability $p_{\text{visit,other}}$ that is higher than p_{visit}. Therefore, we have $p_{\text{visit},m} = p_{\text{visit}}\, p_{\text{visit,other}}$. Note also that the mean number of codewords visited by a single image is $k\, p_{\text{visit}}$. Since p_{visit} increases as the number of local features per image N_{feat} increases, the mean

number of codewords visited increases with both the codebook size k and the number of local features N_{feat}.

If $N_{\text{visit},nm} = n$, then $n\, l_{\text{lda}}$ dimensions are compared between the binary vectors of the two non-matching images. Let $h_{nm,u} = 1$ if the two binary vectors have different signs in the u^{th} dimension and $h_{nm,u} = 0$ otherwise. Suppose $p_{h_{nm,u}}(1) = p_{\text{sign},nm}$. For a non-matching image pair, the variables $h_{nm,u}$ and $h_{nm,v}$ are statistically independent for $u \neq v$. Hence, the Hamming distance $H_{nm} = \sum_{u=1}^{n\, l_{\text{lda}}} h_{nm,u}$ between the two binary vectors can be modeled conditionally as a binomial random variable:

$$p_{H_{nm}|N_{\text{visit},nm}}(h|n) = \binom{n\, l_{\text{lda}}}{h} p_{\text{sign},nm}^{h} \left(1 - p_{\text{sign},nm}\right)^{n\, l_{\text{lda}} - h}. \tag{4.45}$$

For a large set of images, $p_{\text{sign},nm} \approx 0.5$, so on average, the two binary vectors will agree in sign for half of the dimensions.

Similarly, if $N_{\text{visit},m} = n$, then $n\, l_{\text{lda}}$ dimensions are compared between the binary vectors of the two matching images. Again, let $h_{m,u} = 1$ if the two binary vectors have different signs in the u^{th} dimension and $h_{m,u} = 0$ otherwise, and let $p_{h_{m,u}}(1) = p_{\text{sign},m}$. For two matching images, however, the variables $h_{m,u}$ and $h_{m,v}$ are dependent for $u \neq v$. Matching feature descriptors between the two images cause strong correlations between the various dimensions. If the binary residual vectors match in one dimension, then it becomes more likely that they also match in other dimensions. Hence, we cannot use a binomial random variable to accurately model the conditional distribution of the sum $H_m = \sum_{u=1}^{n\, l_{\text{lda}}} h_{m,u}$.

To effectively model the dependence between the different dimensions, we employ a generalized binomial distribution (GDB) [66]. For a GBD, let S_N and F_N denote success and failure, respectively, on the N^{th} Bernoulli trial in the sequence. The probabilities for S_N and F_N depend on the number of successes h in the previous $N - 1$ Bernoulli trials:

$$p\left(S_N | \, h\right) = \left(1 - \pi_m\right) p_{\text{sign},m} + \pi_m \frac{h}{N-1} \tag{4.46}$$

$$p\left(F_N | \, h\right) = \left(1 - \pi_m\right)\left(1 - p_{\text{sign},m}\right) + \pi_m \left(1 - \frac{h}{N-1}\right) \tag{4.47}$$

Hence, the probability for S_N is a mixture of the history-independent a priori probability $p_{\text{sign},m}$ and the history-dependent empirical probability $h/(N-1)$, with a mixing weight π_m. As $\pi_m \to 0$, then the GDB becomes an ordinary binomial distribution. As $\pi_m \to 1$, the dependence between the different Bernoulli trials in the sequence becomes stronger. The probability of h successes in N trials is then defined recursively:

$$\begin{aligned} p_{\text{GBD}}\left(h|N\right) = \quad & p\left(S_N|h-1\right) p_{\text{GBD}}\left(h-1 \,|\, N-1\right) + \\ & p\left(F_N|h\right) p_{\text{GBD}}\left(h|N-1\right). \end{aligned} \tag{4.48}$$

The initial conditions for the GBD are $p_{\text{GBD}}\left(0|1\right) = 1 - p_{\text{visit},m}$ and $p_{\text{GBD}}\left(1|1\right) = p_{\text{visit},m}$. With the GBD, we can model the conditional distribution for H_m:

$$p_{H_m|N_{\text{visit},m}}\left(h|n\right) = p_{\text{GBD}}\left(h \,|\, n\, l_{\text{lda}}\right). \tag{4.49}$$

Since the GBD captures the dependence between Bernoulli trials in the sequence, we can accurately model the dependence between the different dimensions of the residual vector.

Since $C_\theta = n\, l_{\text{lda}} - 2H_\theta$ when $N_{\text{visit},\theta} = n$, the conditional distribution for C_θ is

$$p_{C_\theta|N_{\text{visit},\theta}}\left(c|n\right) = p_{H_\theta|N_{\text{visit},\theta}}\left(0.5\left(n\, l_{\text{lda}} - c\right) \mid n\right) \qquad \theta \in \{nm, m\}. \tag{4.50}$$

Then, the overall distribution for C_θ is a mixture distribution:

$$p_{C_\theta}\left(c\right) = \sum_{n=1}^{k} p_{C_\theta|N_{\text{visit},\theta}}\left(c|n\right) p_{N_{\text{visit},\theta}}\left(n\right) \qquad \theta \in \{nm, m\}. \tag{4.51}$$

For non-matching and matching image pairs, $p_{C_\theta}\left(c\right)$ is a mixture of binomial distributions and a mixture of GBDs, respectively.

Figures 4.10 - 4.14 plot the distributions for C_{nm} and C_m, as the number of codewords k is varied in increments of 60 from $k = 70$ to $k = 310$ and the number of local features $N_{\text{feat}} = 100$ per image is varied in increments of 50 from $N_{\text{feat}} = 100$ to

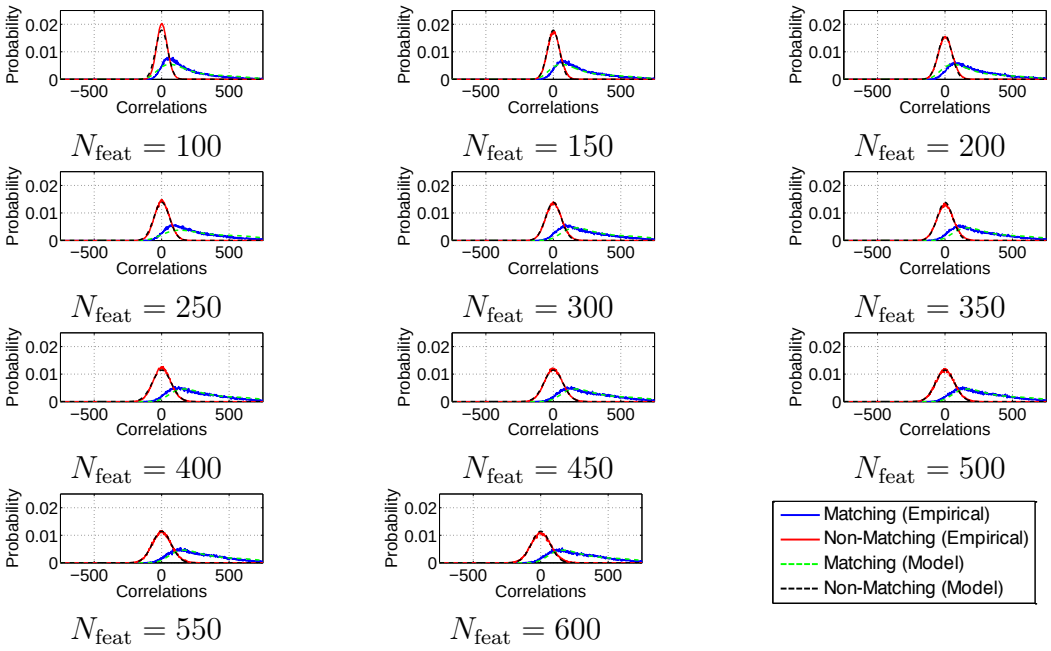


Figure 4.10: Distributions of REVV correlation scores for a codebook size of $k = 70$.

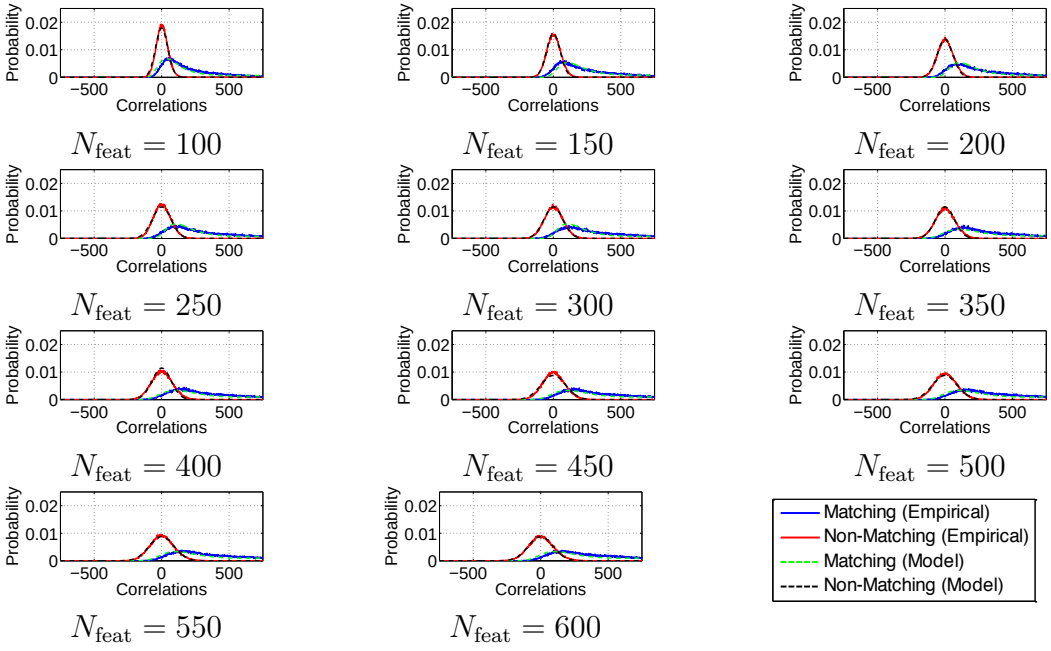


Figure 4.11: Distributions of REVV correlation scores for a codebook size of $k = 130$.

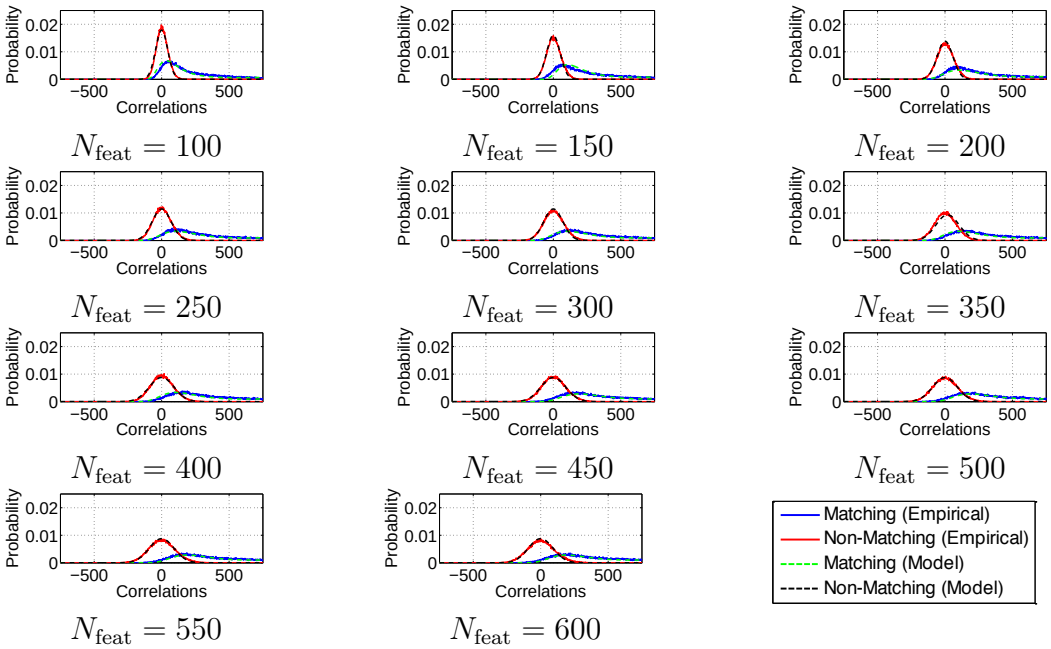


Figure 4.12: Distributions of REVV correlation scores for a codebook size of $k = 190$.

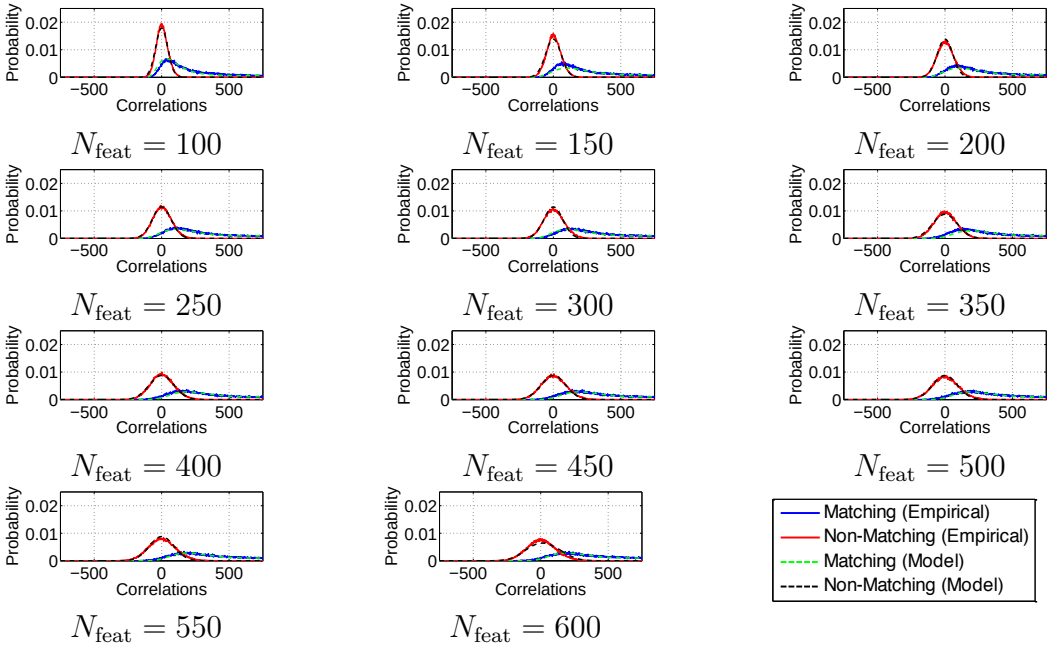


Figure 4.13: Distributions of REVV correlation scores for a codebook size of $k = 250$.

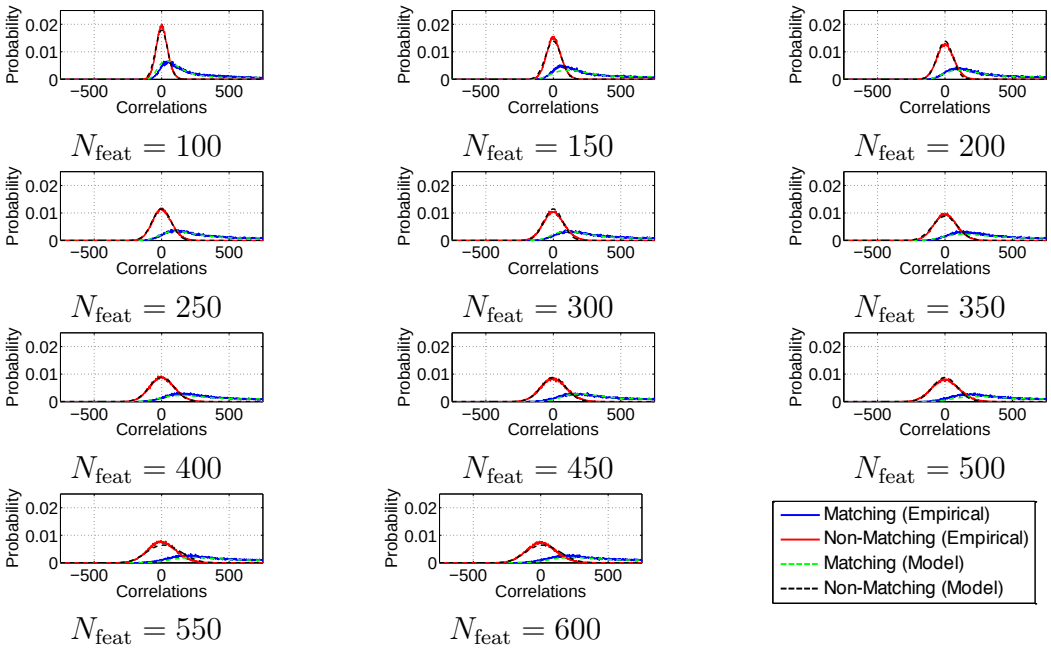


Figure 4.14: Distributions of REVV correlation scores for a codebook size of $k = 310$.

$N_{\text{feat}} = 600$. These are the same sets of parameters tested previously in Section 4.1.7. In each plot, we show the distributions computed according to the statistical model developed in this section next to the empirical distributions computed from sample data. We can observe that the model distributions match well with the empirical distributions. The non-matching distributions are centered at a correlation value of 0 and are well modeled using a mixture of binomial distributions. Due to the dependence between the different dimensions of the residual vector, the matching distributions have a long tail skewed toward large positive correlation values, and these distributions are well modeled with a mixture of GBDs.

A few important characteristics of how the retrieval performance of REVV signatures depends upon k and N_{feat} are revealed by our statistical model:

1. For a fixed codebook size k, the area of overlap between the non-matching and matching distributions changes as N_{feat} increases. At an intermediate value of N_{feat}, the area of overlap between the distributions reaches a minimum, corresponding to the point when the maximal retrieval precision is reached.

2. As the codebook size k increases by 60 codewords each time, the relative change from the set of distributions for one codebook to the set of distributions for the next larger codebook diminishes. The largest change in distributions actually occurs between the codebook with $k = 70$ codewords and the codebook with $k = 130$ codewords. As the distributions change less, the optimal precision value changes less as we increase the codebook size.

3. As the codebook size k increases, the mean number of codewords visited $k\,p_{\text{visit}}$ increases, which means the bitrate required to store each REVV signature in memory increases. When coupled with the second characteristic, there are diminishing gains in the retrieval precision by steadily increasing the memory usage of the database.

These characteristics are entirely consistent with our earlier observations in Section 4.1.7.

4.3.2 Modeling Retrieval Accuracy

When querying a large database of images with REVV signatures, a ranked list of correlation scores is generated. In this list, $N_{db,nm}$ correlation scores $C_{nm}^1, \cdots, C_{nm}^{N_{db,nm}}$ belong to non-matching database images and $N_{db,m}$ correlation scores $C_m^1, \cdots, C_m^{N_{db,m}}$ belong to matching database images. Usually, $N_{db,nm} \gg N_{db,m}$, meaning there are significantly more non-matching images in the database than there are matching images. We assume the scores $C_{nm}^1, \cdots, C_{nm}^{N_{db,nm}}$ and $C_m^1, \cdots, C_m^{N_{db,m}}$ are distributed i.i.d. according to the non-matching correlation model and matching correlation model, respectively, which we have developed in the previous section.

The mean precision at rank 1 (PA1) value measures how often the top-ranked database image is a correct match. Whenever the maximal value of the matching correlation scores $C_m^{\max} = \max\left(C_m^1, \cdots, C_m^{N_{db,m}}\right)$ is larger than the maximal value of the non-matching correlation scores $C_{nm}^{\max} = \max\left(C_{nm}^1, \cdots, C_{nm}^{N_{db,nm}}\right)$, then the top-ranked database image is correct. The cumulative distribution function (CDF) for

$C_\theta^{\max}$ is found to be

$$\begin{aligned} F_{C_\theta^{\max}}(c) &= P\left(C_\theta^{\max} \le c\right) && (4.52)\\ &= P\left(C_\theta^1 \le c, \cdots, C_\theta^{N_{db,\theta}} \le c\right) && (4.53)\\ &= P\left(C_\theta^1 \le c\right)^{N_{db,\theta}} && (4.54)\\ &= \left(F_{C_\theta^1}(c)\right)^{N_{db,\theta}} \qquad \theta \in \{nm, m\} && (4.55) \end{aligned}$$

where the third line follows from the i.i.d. assumption for the correlation scores. Subsequently, the probability mass function (PMF) for $C_\theta^{\max}$ can be obtained by taking discrete differences of the CDF. Now, the PA1 value is predicted to be

$$\begin{aligned} \text{PA1} &= P\left(C_m^{\max} \ge C_{nm}^{\max}\right) && (4.56)\\ &= \sum_c P\left(c \ge C_{nm}^{\max}\right)\, p_{C_m^{\max}}(c) && (4.57)\\ &= \sum_c F_{C_{nm}^{\max}}(c)\, p_{C_m^{\max}}(c)\,. && (4.58) \end{aligned}$$

In Figure 4.15, we plot the PA1 value versus the bitrate for the MPEG CDVS Dataset, with a range of different numbers of codewords and different numbers of local features. Since the mean number of codewords visited by an image is $k\,p_{\text{visit}}$, the mean bitrate associated with each PA1 value is $k\,(p_{\text{visit}}\,l_{\text{lda}} + 1)$ bits per image. The solid curves represent the empirical measurements, while the dashed curves represent the values predicted by our statistical model. Because we ignored second-order effects such as the correlation weighting and the score normalization when developing our model, our model curves do not exactly match the empirical curves. However, our model curves capture well the two most important characteristics: (i) there is an optimal number of local features $N_{\text{feat}}^*(k)$ to choose for each codebook size k that yields the maximal PA1 value $P^*(k)$, and (ii) as the codebook size k increases, the gaps between the maximal PA1 values decrease, so there are rapidly diminishing gains to using larger codebooks. Our model can be used to select the optimal pair of k and N_{feat} values in the REVV-based retrieval system for any given database memory budget.

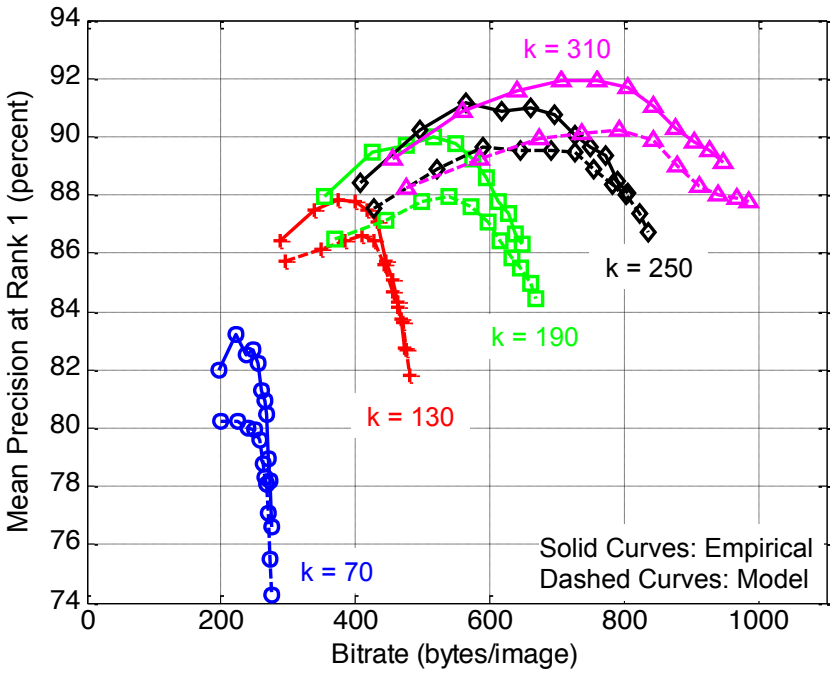


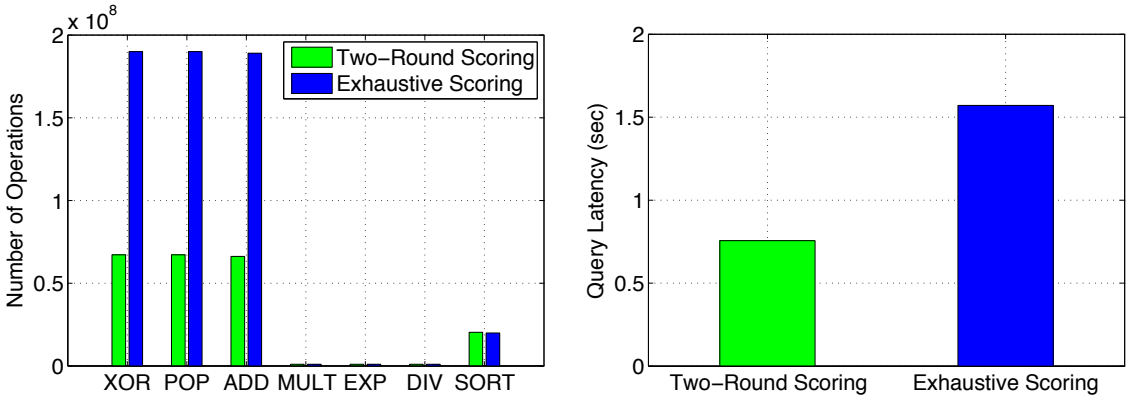
Figure 4.15: Mean precision at rank 1 (PA1) versus the bitrate in bytes/image, for query images and 20K database images in the MPEG CDVS Dataset. The empirical precision values (solid lines) are plotted next to the model precision values (dashed lines).

4.4 Coding and Retrieval Results

This section reports the large-scale coding and retrieval results for REVV. The codebook, global PCA transform, cell-specific LDA transform, and correlation weights for REVV are all generated from the training dataset described in Appendix C. We have selected the following parameters because they yield the best retrieval accuracy when applied to SIFT features.

- For the power law transformation, we use $\alpha = 0.5$.
- For the global PCA, we use $l_{\text{gpca}} = 64$ eigenvectors.
- For the cell-specific LDA, we use $l_{\text{lda}} = 32$ eigenvectors per codeword.

Although the REVV codebook size could easily be adjusted to accomodate different memory budgets, as shown in Section 4.1.7, we choose a codebook size of $k = 190$



(a) (b)

Figure 4.16: Computational complexity of REVV signature comparisons for two-round scoring and exhaustive scoring algorithms, in terms of (a) total number of various operations in searching a database for a single query, and (b) total latency in seconds for a single query. The parameters for the scoring algorithms are $N_{db} = 1$M images, $k = 190$ codewords, $\Delta_{r1} = 3$, and $N_{r2} = 25$K images. Latencies are measured on a single Intel Xeon 2.4 GHz processor.

codewords for the following comparisons against other compressed database methods. For this codebook size, both our analysis in Section 4.3 and empirical measurements show that the optimal number of local features per image is $N^*_{\text{feat}}(190) = 250$ SIFT features. These 250 SIFT features are selected for each image using the same feature selector [76] that was used in Chapter 3.

First, we show that the multi-round scoring algorithm presented in Section 4.2.3 is as accurate as an exhaustive scoring algorithm, but requires a fraction of the computational cost. Figure 4.16(a) plots the number of operations required to search a database of $N_{db} = 1$M images for a two-round scoring algorithm versus an exhaustive scoring algorithm. The operations are computed using the expressions listed in Table 4.2. For two-round scoring, we set the parameters so that only 1 in every $\Delta_{r1} = 3$ codewords are visited in the first round and only $N_{r2} = 0.025 \cdot N_{db} = 25$K database images are scored in the second round. The number of operations in the database search are significantly reduced by using two-round scoring. This translates into significant savings in the query latency, as shown in Figure 4.16(b). The latency is averaged over all 8,313 query images in the MPEG CDVS Dataset. The average latency per

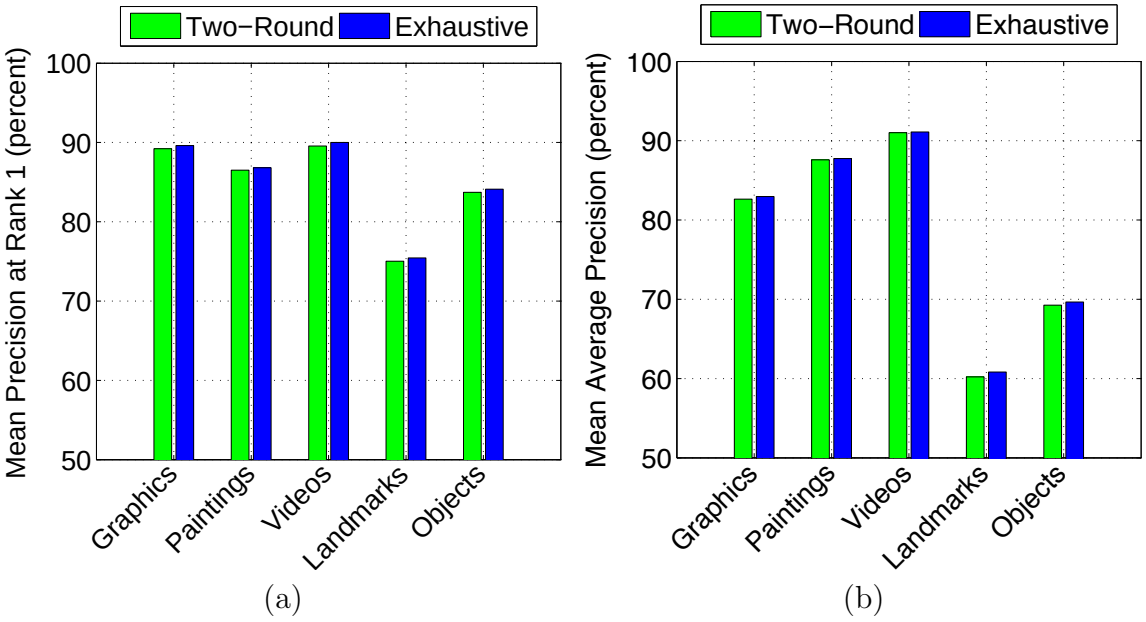


Figure 4.17: Retrieval accuracy for REVV signatures with two-round scoring and exhaustive scoring algorithms on the MPEG CDVS Dataset, in terms of (a) mean precision at rank 1 (PA1) and (b) mean average precision (MAP). The parameters for the scoring algorithms are $N_{db} = 1$M images, $k = 190$ codewords, $\Delta_{r1} = 3$, and $N_{r2} = 25$K images.

query is reduced from about 1.65 seconds to 0.75 seconds. The high retrieval accuracy achieved with REVV is well preserved when switching from the exhaustive scoring scheme to the two-round scoring scheme. Figure 4.17 plots the retrieval accuracy on the MPEG CDVS Dataset in terms of PA1 and MAP values. Both two-round scoring and exhaustive scoring yield almost identical precision measurements across the five categories of the MPEG CDVS Dataset, with two-round scoring being only a fraction of a percent lower on average.

Next, we compare the REVV signature against several competing methods:

- Raw: This refers to a database constructed from uncompressed tree histograms. A vocabulary tree with depth $d = 6$ and $k = 10$ is employed, because Section 3.4 showed that this large tree obtained the best retrieval accuracy and lowest query latency. Also, soft binning with $m = 3$ is used to increase the retrieval accuracy.

- THC: This refers to the tree histogram coding method described in Section 3.2. The same vocabulary tree as the Raw baseline method is used here. For entropy coding, we select the RBUC codec described in Section 3.2.3 because it offers good compression efficiency and low decoding delays.

- IIC: This refers to the inverted index coding method described in Section 3.3. All parameters are the same as for THC, except that the inverted index is compressed rather than the tree histograms.

- SCFV: This refers to the scalable compressed fisher vector [125, 124]. Similar to REVV, SCFV aggregates the quantization residuals in a Fisher vector representation. We compare against a version of SCFV with $k = 128$ codewords that uses the same amount of memory as REVV with $k = 190$ codewords.

Figure 4.18 plots the PA1 and MAP values across the five categories of the MPEG CDVS Dataset for THC, IIC, SCFV, and REVV. The accuracies for REVV and THC/IIC are very similar across the five categories, although REVV outperforms THC/IIC by several percent in the objects category. Across all five categories, REVV consistently outperforms SCFV by several percent. Although REVV and SCFV both use feature residuals, REVV utilizes an improved score normalization, correlation weighting, dimensionality reduction, and optimization of the number of local features.

Figure 4.19 plots the memory usage for the 1M database images in the MPEG CDVS Dataset, where the memory usage is separated into two types: (i) the memory used by the database signatures and (ii) the memory usage used by auxiliary data. For Raw, THC, and IIC, the database signatures are the tree histograms, while the auxiliary data consists of (i) the vocabulary tree with branch factor $k = 10$ and depth $d = 6$, where each node is a 128-dimensional vector, (ii) the IDF weights for the leaf nodes of the vocabulary tree, and (iii) the probability tables for the identifier runlength and feature count distributions. For REVV, the database signatures are the binarized REVV signatures, while the auxiliary data consists of (i) a codebook of visual words, (ii) global PCA and cell-specific LDA eigenvectors, and (iii) a table of correlation weights. Similarly for SCFV, the database signatures are the binarized

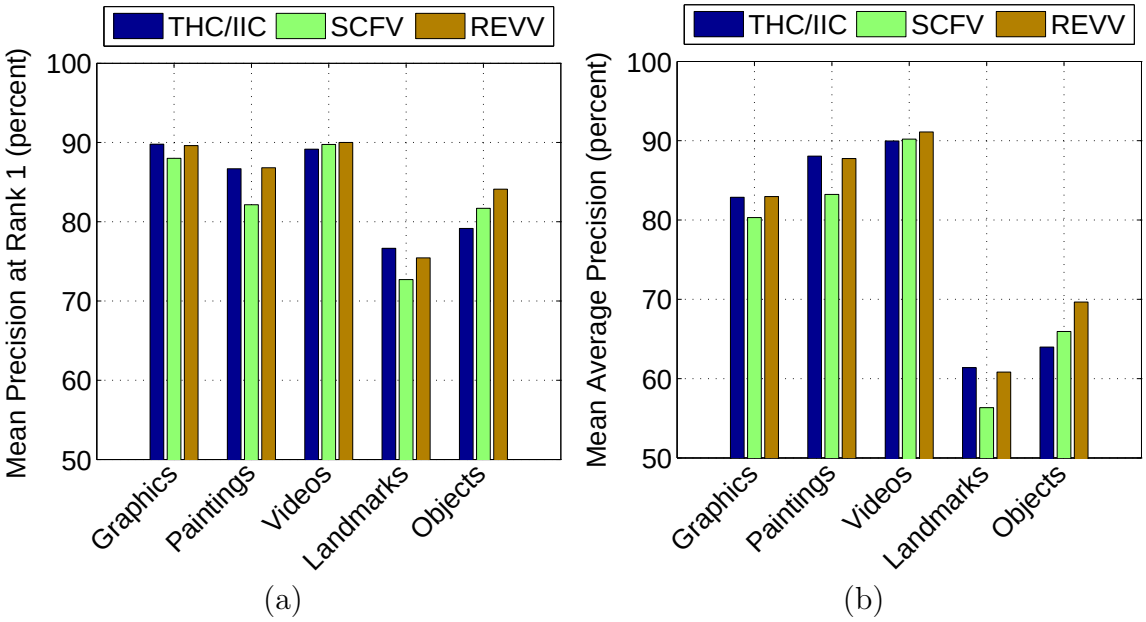
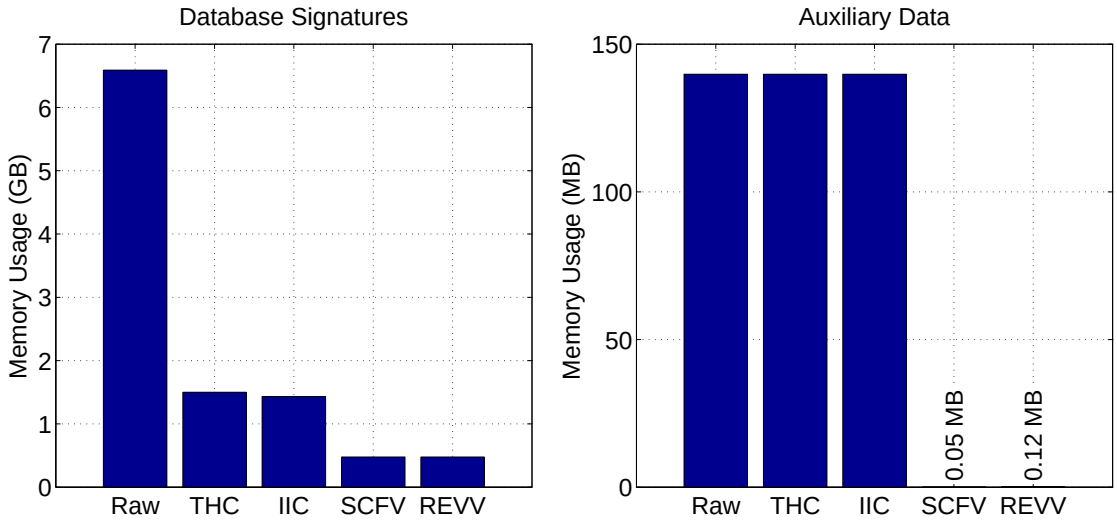


Figure 4.18: Retrieval accuracy for several different database compression methods on the MPEG CDVS Dataset, in terms of (a) mean precision at rank 1 (PA1) and (b) mean average precision (MAP). Raw, THC, and IIC use a vocabulary tree with depth $d = 6$, branch factor $k = 10$, and soft binning $m = 3$. SCFV and REVV use codebooks of $k = 128$ and $k = 190$, respectively.

SCFV signatures, while the auxiliary data consists of (i) a codebook of visual words, (ii) global PCA eigenvectors, and (iii) a table of correlation weights.

From Figure 4.19, we can observe the major advantage of the features residual methods. First, Figure 4.19(a) shows that the feature residual REVV and SCFV methods significantly reduce the memory usage of the database signatures compared to the feature histogram THC and IIC methods. Second, Figure 4.19(b) shows the large savings in the memory usage of the auxiliary data achieved by REVV or SCFV. One of our primary goals mentioned at the beginning of the chapter was to eliminate the large codebooks required by feature histogram methods while achieving the same high retrieval accuracies. Now, with feature residual methods like REVV and SCFV, we can achieve the same retrieval accuracy as THC and IIC, while using a codebook that is several orders of magnitude smaller.



(a) (b)

Figure 4.19: Memory usage for several different database compression methods on the MPEG CDVS Dataset. The memory usage is divided between (a) database signatures and (b) auxiliary data. The RBUC codec is used for entropy coding of tree histogram symbols.

4.5 Summary

In this chapter, we have developed a new method for building a memory-efficient database using feature residuals. These residuals were previously ignored and discarded when we generated feature histograms in Chapter 3. Now, we have shown that the residuals contain valuable information about local neighborhood feature statistics, which can be exploited to build a compact global image signature. Our contributions culminated in the formation of the residual enhanced visual vector (REVV). Compact REVV signatures are capable of (i) accurate search of a large visual database, (ii) memory-efficient indexing of a large collection of images, and (iii) fast comparisons directly in the compressed domain.

The REVV signature is constructed by a carefully designed and optimized sequence of simple operations: nonlinear mapping with power law, global linear transform, quantization, residual aggregation and normalization, cell-specific linear transform, and signed binarization. Once constructed, a query REVV signature can be

effectively compared against a database of REVV signatures using weighted correlations, where the weights are designed to favor observations that are more likely to have originated from matching image pairs. The correlations require just a small number of XOR and POPCNT operations and table lookups. Furthermore, the database search can be made significantly faster with a multi-round non-exhaustive scoring algorithm without degrading the final retrieval accuracy. To help explain important characteristics in REVV-based retrieval and optimize the signature parameters, we have also performed a thorough statistical analysis of the REVV signature's retrieval performance. We have carefully modeled the correlation scores for matching and non-matching images and modeled the retrieval ranking process. Our analysis reveals how the retrieval accuracy can be maximized by choosing an optimal pair of the number of codewords and number of local features for some given memory budget. Lastly, in the experimental results, we have compared REVV against several other global signatures and demonstrated noticeably better performance in terms of retrieval accuracy and memory usage.

By its core design, REVV signatures significantly reduce the memory usage of the database signatures and the auxiliary data compared to the compressed feature histograms from Chapter 3. The large vocabulary tree required by feature histograms to achieve high retrieval accuracies can be conveniently replaced by a small codebook in the case of REVV. As a result, a REVV-based retrieval engine is easier to deploy on mobile devices with small memory capacities, or multiple REVV-structured databases can be operated in parallel for different visual search applications and share the same memory space on a single server. In the next chapter, we demonstrate some practical applications of a memory-efficient REVV-structured database in building robust, low-latency MVS systems.

Chapter 5

Mobile Visual Search Applications

For accurate object recognition in mobile visual search (MVS) systems, images captured by the mobile device's camera are usually compared against a database of labeled images. When one or more similar database images are found, the labels for these database images can be used to describe the recognized object(s) as well as to retrieve their relevant metadata for the user. We desire a near real-time system response in order to provide continuous augmentations in the mobile device's viewfinder. If the database exists on a remote server that must be contacted on every new query, then the system is vulnerable to fluctuations in the network speed and the server computational load. Slow network transmissions or server congestion can severely degrade the user experience.

In this chapter, we explain how a memory-efficient database of image signatures stored on a mobile device can enable fast local queries.[1] By using local database search, we can achieve low query latencies anywhere and anytime, independent of unpredictable external network or server conditions. To realize this goal, the image signatures must be extremely compact to fit in the small memory capacity of a mobile device, capable of efficient comparisons across a large database, and robust to the challenging visual distortions typically encountered in MVS queries. In Section 5.1, we leverage the memory-efficient databases that we have previously developed to build a new on-device image matching system. In particular, we show that residual

[1] Preliminary results of our work have appeared in [44, 42, 36].

Symbol	Meaning
$\mathbf{x}^t$	RIFF keypoint location in frame t
$\mathbf{v}^t$	RIFF descriptor in frame t
b	Spatial block size for RIFF tracker
t_{L_1}	Threshold for L_1 norm between matching RIFF descriptors
N_{track}	Number of RIFF features tracked between two frames
t_{high}	High threshold in motion classification
t_{low}	Low threshold in motion classification
$\mathbf{R}_t$	Original REVV signature for frame t
$\mathbf{S}_t$	Interframe-coded REVV signature for frame t
$u_{t,i}$	Binary indicator of whether codeword i is visited in frame t
$\mathbf{b}_{t,i}$	Binary residual vector at codeword i in frame t
$r_k(t, t-1)$	Interframe codeword similarity between frames t and $t-1$
t_{r_k}	Threshold for interframe codeword similarity
t_{RANSAC}	Threshold for number of RANSAC inliers in local search
R_{Indep}	Uplink bitrate for independent coding of REVV
R_{SCP}	Uplink bitrate for interframe coding of REVV with SCP
R_{SFP}	Uplink bitrate for interframe coding of REVV with SFP
$R_{\text{SFP+LS}}$	Uplink bitrate for interframe coding of REVV with SFP+LS
ΔR_{SCP}	Bitrate savings for SCP compared to independent coding
ΔR_{SFP}	Bitrate savings for SFP compared to independent coding
$\Delta R_{\text{SFP+LS}}$	Bitrate savings for SFP + LS compared to independent coding
N_{Frames}	Number of frames per second
N_{DF}	Number of detection frames per second
N_{FPF}	Number of forward propagation frames per second
$\text{PA1}_{\text{remote}}$	Mean precision at rank 1 for remote database matching system
$\text{PA1}_{\text{hybrid}}$	Mean precision at rank 1 for hybrid image matching system
p_{local}	Probability that a local database search is successful

Table 5.1: Mathematical symbols used in Chapter 5.

enhanced visual vector (REVV) signatures are highly effective in constructing an on-device database with a small memory footprint. As a key part of this system, we also develop a motion-adaptive query selection mechanism to automatically infer changes in user interest from the motion of the mobile device and select high-quality query frames for database search.

With a local database stored on the mobile device, a question naturally arises regarding how this database is updated over time. Section 5.2 develops a hybrid image matching system that simultaneously performs efficient query expansion and local database update. A compact REVV signature is sent from the mobile device to the server to search a larger REVV-structured database stored on the server. In response, the server can send back the labels, metadata, REVV signatures, and local features for the top-ranked database candidates in the downlink. When the mobile device receives this data from the server, the local database will be updated to improve the accuracy of the current query and future queries. For the reader's reference, all mathematical symbols used in this chapter are listed in Table 5.1.

5.1 On-Device Image Matching System

Figure 5.1 shows an overview of our on-device image matching system. There are three major processing stages for each query:

- Motion-adaptive query selection: The query is initiated by analyzing the motion between viewfinder frames and detecting periods of low motion. During each low-motion interval, one or more query frames are selected.
- Local database search: Local image features are extracted from the query frames, REVV signatures are first generated from the local features and then matched against a local REVV database, and geometric verification is performed on a shortlist of selected database candidates.
- Viewfinder augmentation: The recognized objects are tracked in the viewfinder for as long as the user maintains focus on these objects. The viewfinder is augmented with important information about the recognized objects.

The user interface for our system is designed to be very intuitive. All the user has to do is point the mobile device's camera at objects in the scene, and our system will recognize these objects and augment the viewfinder with information about these objects within 1 second. The following few sections provide the important technical details about how each of the three major processing stages are designed and developed.

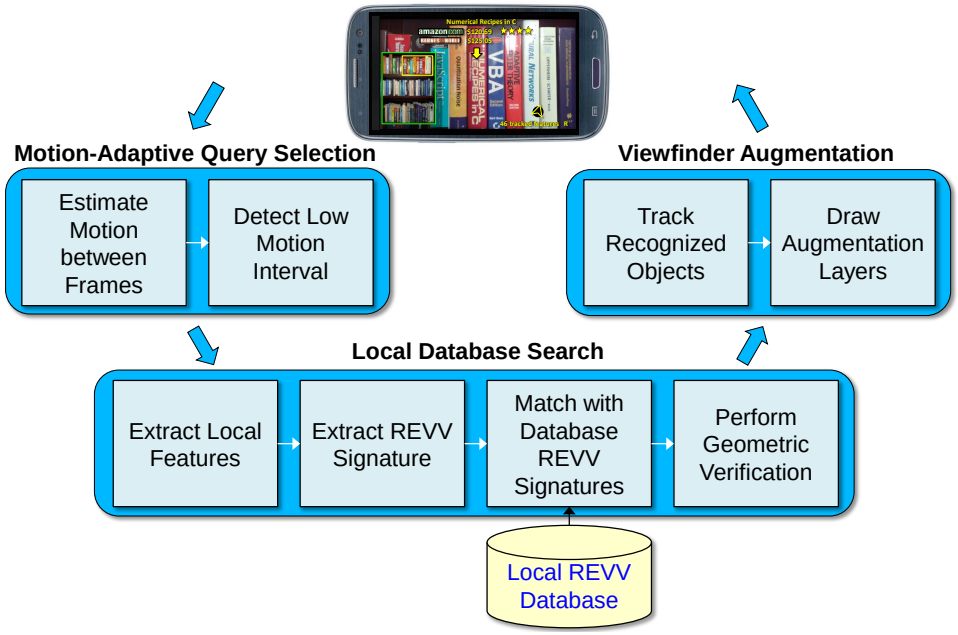


Figure 5.1: On-device image matching system with three major processing stages: (i) motion-adaptive query selection, (ii) local database search with an on-device database of REVV signatures, and (iii) viewfinder augmentation.

5.1.1 Motion-Adaptive Query Selection

Most traditional tracking algorithms, such as the Kanade-Lucas-Tomasi (KLT) tracker based on optical flow [191], are too slow to run at video frame rates on a mobile device. Hence, we employ a low-complexity tracker based on RIFF features [203] that is designed for efficient operation on mobile platforms. In this tracker, up to 100 RIFF features are extracted for each frame and matched with RIFF features of the previous frame. Each RIFF feature in the current frame t has a location $\mathbf{x}^t = (x^t, y^t)$ and an l-dimensional descriptor $\mathbf{v}^t \in \mathbb{R}^l$. This RIFF feature in frame t is considered to be matched to a RIFF feature in the previous frame $t-1$ with location $\mathbf{x}^{t-1} = (x^{t-1}, y^{t-1})$ and descriptor $\mathbf{v}^{t-1}$, if $\mathbf{x}^t$ and $\mathbf{x}^{t-1}$ fall into neighboring $b \times b$ spatial bins and $\|\mathbf{v}^t - \mathbf{v}^{t-1}\|_1$ is less than a threshold t_{L_1}. In this case, the RIFF feature is considered to be successfully tracked from frame $t-1$ to frame t. From the research results of optimizations to the RIFF tracker [201], the tracking error is minimized by setting the spatial bin size to $b = 8$ and the threshold to $t_{L_1} = 2.4$, so

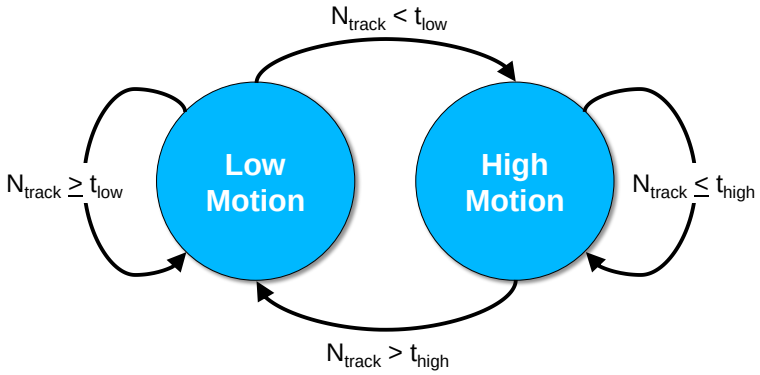


Figure 5.2: Low motion and high motion states for viewfinder frames. Transitions between the two states depend on the number of tracked features N_{track} between consecutive viewfinder frames.

these recommended parameter settings are used in our current system.

Let N_{track} denote the number of RIFF features tracked between two consecutive frames, according to the criteria mentioned above. When the user moves the mobile device slowly, N_{track} is high because a large number of local keypoints can be consistently detected and tracked. Conversely, when the user moves the mobile device rapidly, the number of tracked features drops precipitously and often to 0, because motion blur and large displacements lead to a failure to detect or track most of the local keypoints. Based on this observation, we develop the two-state motion classifier shown in Figure 5.2. When the system is in the low-motion state, it remains in that state as long as N_{track} stays above a threshold t_{low}. Similarly, when the system is in the high-motion state, it remains there as long as N_{track} stays below a different threshold t_{high}. Note that $t_{\text{low}} < t_{\text{high}}$ always. The use of two thresholds guards against rapid transitions between the two motion states caused by small amounts of noise and makes the system more stable. The gap $t_{\text{high}} - t_{\text{low}}$ is adjusted to be slightly higher than the standard deviation of N_{track} within each low-motion interval.

To see the effect of this motion classifier in practice, we present the results of the classifier on two different viewfinder sequences in Figures 5.3 and 5.4. Each sequence has a frame rate of 30 frames/second. First, Figure 5.3(a) and Figure 5.4(a) show some keyframes taken from each sequence. Then, Figure 5.3(b) and Figure 5.4(b)

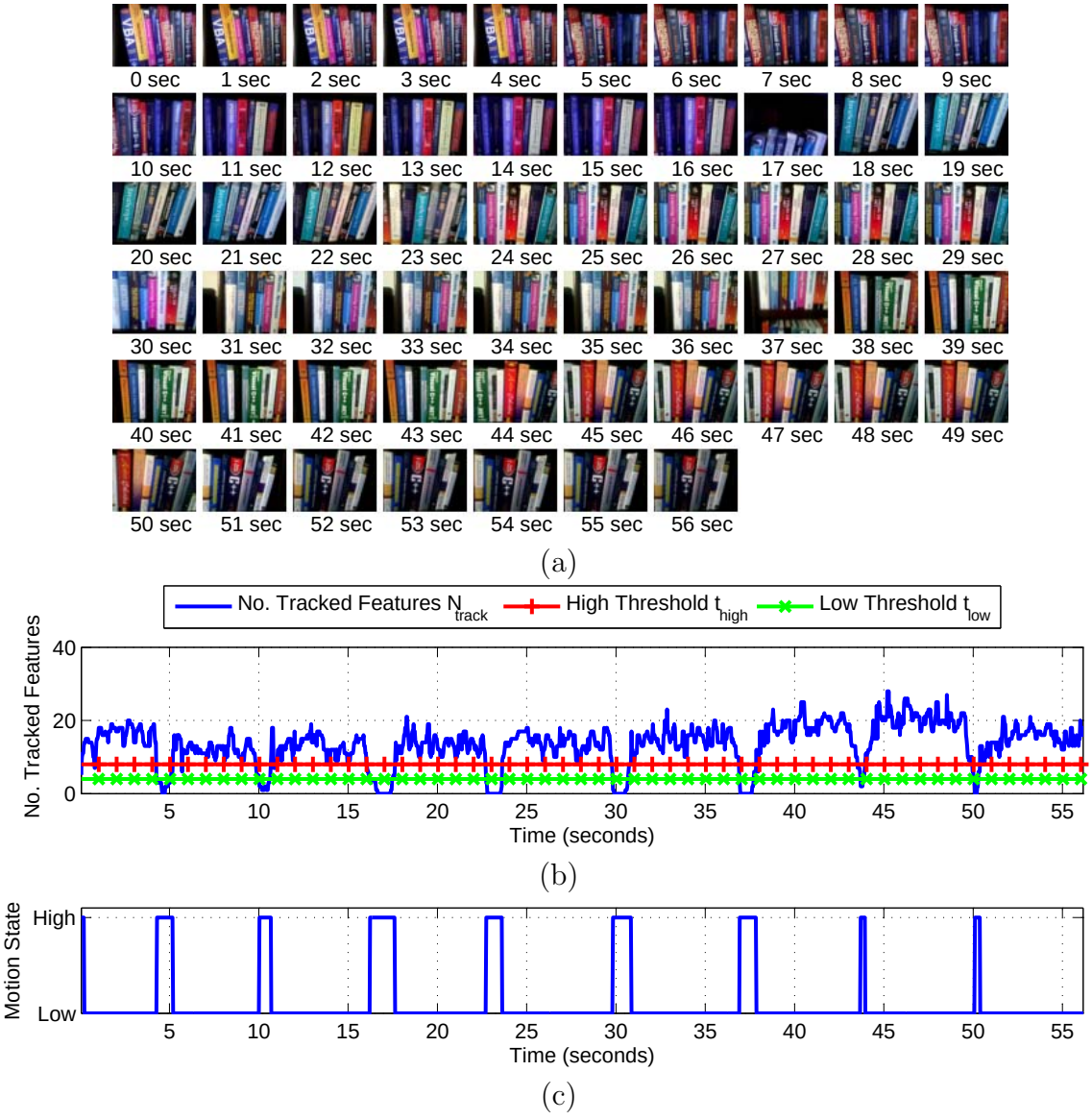


Figure 5.3: First motion-adaptive query selection example, showing (a) keyframes of viewfinder sequence, (b) number of tracked features, and (c) motion classification.

show plots of the number of tracked features N_{track} over time. Finally, Figure 5.3(c) and Figure 5.4(c) show the corresponding motion classifications determined by our algorithm. The system enters the high-motion state whenever N_{track} drops sharply to below the threshold t_{low}. There is a clear correspondence between the valleys in Figure 5.3(b) and Figure 5.4(b) and the high-motion intervals in Figure 5.3(c) and

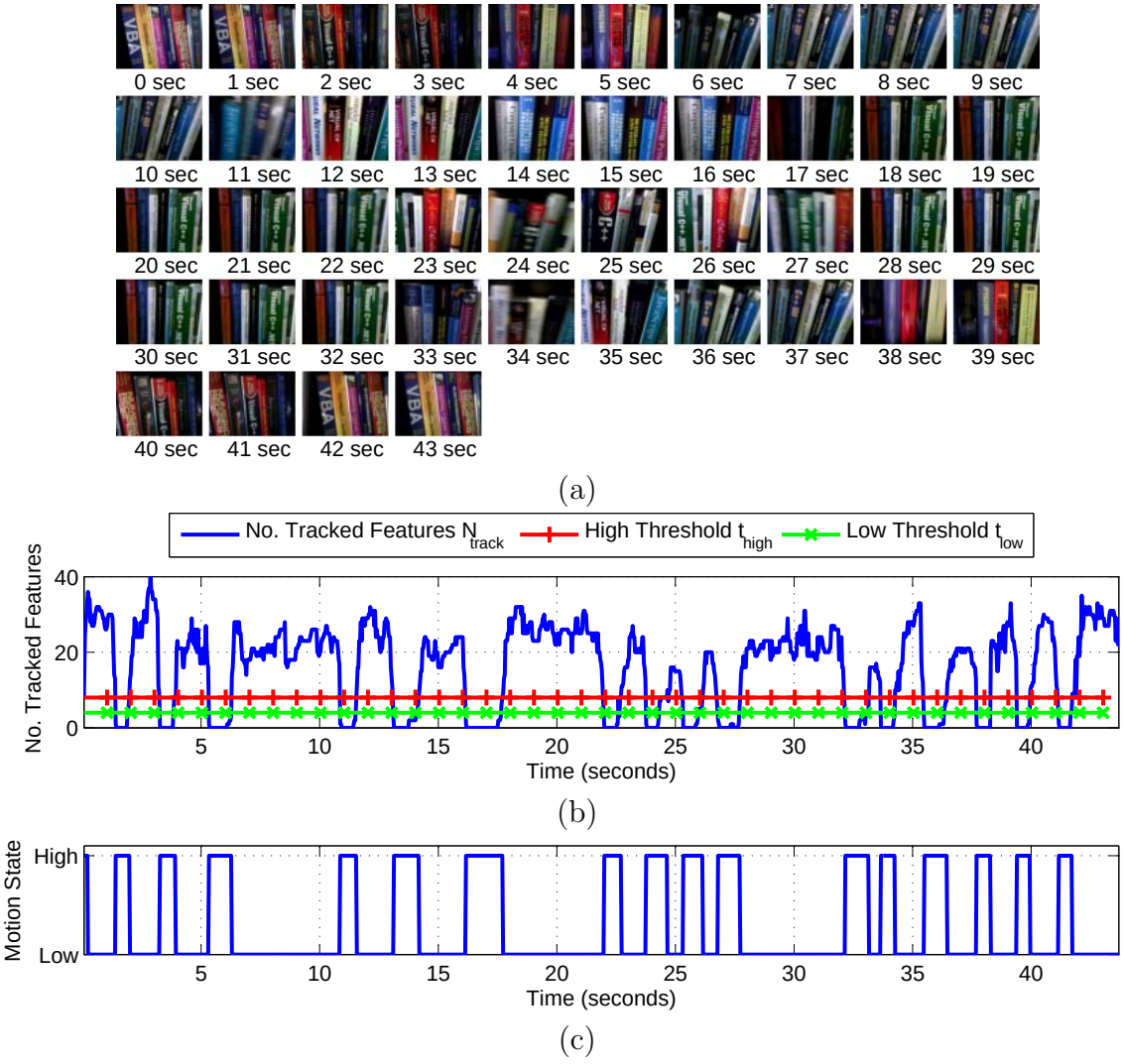


Figure 5.4: Second motion-adaptive query selection example, showing (a) keyframes of viewfinder sequence, (b) number of tracked features, and (c) motion classification.

Figure 5.4(c). The importance of the two thresholds also becomes apparent. Since N_{track} can vary significantly over time, a single threshold would cause rapid oscillations between the two motion states during various parts of the viewfinder sequence.

With the motion classification result, we can now automatically determine how our system should respond to external changes in user interest and scene contents. At the beginning of every low-motion interval, which corresponds to a period where the

Figure 5.5: (Top row) Viewfinder frames taken from low-motion intervals. (Bottom row) Viewfinder frames taken from high-motion intervals. The low-motion frames have clear and crisp details, but the high-motion frames suffer from motion blur.

user is very likely interested in what is shown in the viewfinder, we select one or more viewfinder frames to start a new query. Then, as long as we stay in that low-motion interval, the recognized objects are tracked and augmentation information is drawn in the viewfinder. As the user shifts attention to other objects in the scene, the mobile device will move and a high-motion interval will be detected corresponding to the transitional period. With this motion-adaptive query selection mechanism, the user never has to press a button to initiate a query. Instead, recognition results magically appear in the viewfinder almost instantly after a user focuses on some objects in the scene.

There is yet another important benefit provided by motion-adaptive query selection. Figure 5.5 shows a set of viewfinder frames selected from low-motion intervals in the top row and a set of viewfinder frames selected from high-motion intervals in the bottom row. Low-motion frames have very clear and crisp image features, so high-quality local features will be extracted from these frames and will lead to accurate retrieval results. Conversely, the high-motion frames are degraded severely by motion blur, which affects both the effectiveness of the keypoint detection and the descriptor computation operations. As a result, low-quality local features will be extracted from these blurry frames and yield poor retrieval results. Motion-adaptive

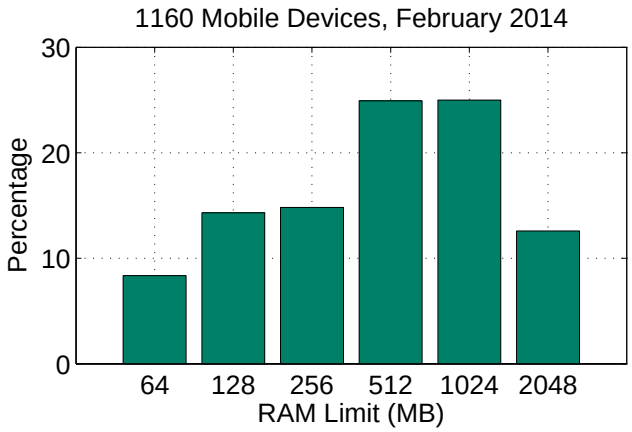


Figure 5.6: Histogram of RAM limits for 1,160 mobile devices available on the market in February of 2014.

query selection enables us to select only the high-quality clear frames and entirely avoid the low-quality blurry frames.

An alternative sampling method that ignores the motion information could be a uniform temporal sampling of keyframes. Uniform sampling will have a lower average retrieval accuracy than motion-adaptive query selection due to periodic selection of the blurry frames. Uniform sampling will also waste battery power, by processing these blurry frames through the entire on-device image matching pipeline, even though these low-quality frames are unlikely to yield useful retrieval results. Motion-adaptive query selection avoids both of these problems by intelligently deciding when to generate a query and trigger the image matching process.

5.1.2 Local Database Search

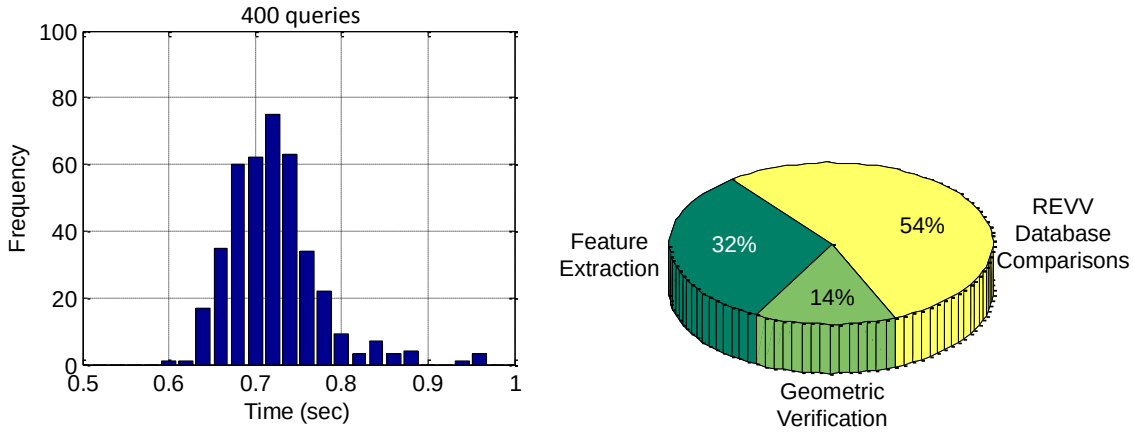
To search a local on-device database, we must first be able to store the database in the small amount of random access memory (RAM) available on the mobile device. Figure 5.6 plots a histogram of the RAM limits on 1160 different mobile devices, which are available on the market as of February 2014. Table 5.2 lists the RAM usage for a database of 100K images, where the database images are represented by the compact REVV signatures. These 100K images could represent images of all products in a supermarket or bookstore, images of all landmarks in the local vicinity

Data	Memory Usage
Database Signatures	49 MB
Auxiliary Data	0.12 MB

Table 5.2: Memory usage for an on-device database of 100K images, where REVV signatures compactly represent the database images.

as estimated from the current GPS coordinates, or images in the personal photo collections of a group of friends or colleagues, amongst many possible examples. In total, less than 50 MB of RAM are required for the entire database, so this database can be stored on essentially 100 percent of mobile devices on the market today. For the majority of mobile devices, the 50 MB of RAM used by our compact database represents only a small fraction of the overall RAM capacity, so other applications running concurrently on the same device have most of the RAM still available. The small amount of memory usage also enables the database to be quickly transferable from the SD card into RAM when the MVS application is initially launched on the mobile device or when the MVS application switches from one database to another.

After motion-adaptive query selection initiates a new query, 250 SURF features [20, 18] are extracted from each selected VGA-resolution viewfinder frame. We use the extended 128-dimensional SURF descriptor, because it is more discriminative than and requires about the same amount of time to compute as the shorter 64-dimensional SURF descriptor [20, 18]. Then, a REVV signature is generated from the SURF feature set of each image using the generation pipeline described in Section 4.1 The current version of REVV uses a codebook of $k = 190$ codewords, $l_{\text{pca}} = 64$ global PCA eigenvectors, and $l_{\text{lda}} = 32$ cell-specific LDA eigenvectors per codeword. After the query REVV signature is compared against the database REVV signatures using the fast multi-round scoring algorithm described in Section 4.2, a ranked list of the selected database candidates is created. The top 25 database images in this ranked list are further compared to the query frame(s) using a distance ratio test [134] and RANSAC with an affine model [71]. The affine model estimated by RANSAC is also used later on in the viewfinder augmentation stage to estimate the location of the recognized objects in the viewfinder frames.



(a) (b)

Figure 5.7: Measurements of on-device image matching latency for 400 different queries against a database of 100K images. The times are measured on a Samsung Galaxy S3 smartphone with a 1.4 GHz processor. (a) Histogram of latencies. (b) Percentage of time spent in feature extraction, database search, and geometric verification.

To create a highly interactive user experience, the image matching latency must be very low. We have implemented the on-device image matching system illustrated in Figure 5.1 on a Samsung Galaxy S3 smartphone. This phone has a 1.4 GHz processor and 1 GB of RAM. Figure 5.7(a) plots a histogram of the image matching latency for 400 different queries executed by our system. On average, each query takes about 0.7 seconds, which enables near real-time augmentations to appear in the viewfinder. Figure 5.7(b) plots the percentage of time spent in local feature extraction, database comparisons with REVV signatures, and geometric verification of the top database candidates. Because we are searching a large database of 100K images on the mobile device, the REVV database comparisons take about half of the database search time. Searching a smaller on-device database will reduce the REVV comparison latency. The feature extraction and geometric verification latencies are independent of the database size, assuming the shortlist length for geometric verification is fixed. Importantly, the quick response times shown in Figure 5.7(a) can be achieved anywhere and anytime, independent of external network or server conditions, because only fast on-device image matching is carried out.

Figure 5.8: Examples of query frames showing a stack of books being segmented into individual book spines. The yellow lines denote the segmentation boundaries determined by the algorithm in [41, 43].

To improve the recognition accuracy on top of the general image matching techniques we have already described, we can additionally make certain category-specific enhancements. Such enhancements require some additional prior knowledge of what type of object is searched, e.g., as indicated by the user when the application is first launched. For book spine recognition, the query frame contains multiple book spines which can act as clutter or distractions toward one another. Thus, we perform a segmentation of the query frame into individual book spines and then query each segmented spine against a database of individual book spines [41, 43]. Figure 5.8 shows two examples of query frames being segmented into individual book spines. For outdoor landmark recognition, many existing image collections including streetview images and social media images have geotags associated with the database images. Hence, the GPS transceiver on a mobile device can be used to coarsely estimate the device's current location and constrain the database search. Although the mobile device's GPS estimate is often noisy and inaccurate, particularly in urban canyons, the estimate can still help in limiting the image comparisons to only the database images in the vicinity of the mobile device. This geographical constraint results in both an improvement in the retrieval accuracy and a reduction in query latency [202, 35].

Media Cover Recognition: We augment the viewfinder with the title, user rating, and competing prices of each recognized media cover; we highlight the boundary of the media cover; and we play a related audio sample, e.g., a song for a CD cover or the audio track of a scene for a DVD cover.

Book Spine Recognition: We augment the viewfinder with the title, user rating, and competing prices of each recognized book spine; we highlight the location of the books in a thumbnail of the larger bookshelf (lower left corner); and we generate a text-to-speech reading of a summary for each book.

Outdoor Landmark Recognition: We augment the viewfinder with the name, address, and main phone number of the recognized outdoor landmark. In the lower right corner, there is a clickable map of the local neighborhood, centered on the recognized landmark, which can be enlarged to see more details.

Figure 5.9: Viewfinder augmentation examples for recognized media covers, book spines, and outdoor landmarks.

5.1.3 Viewfinder Augmentation

Once the objects in the query frame are recognized, we can augment the viewfinder with the important information about the objects. Figure 5.9 shows three augmentation examples generated by our system for media covers, book spines, and outdoor landmarks, where the chosen augmentation information is adapted to the category of the recognized object. For all three examples, the title and important metadata about each recognized object is drawn in the viewfinder on top or in the near vicinity of the

object. In addition to the visual augmentations, the user's perception of the object can also be augmented by audio clips that we retrieve and play for each recognized object, such as song on a CD album or a text-to-speech reading of a book's summary.

The same RIFF-based tracker that provides constant monitoring of the object motion states for motion-adaptive query selection now serves a second useful purpose. Each time the tracker propagates a set of RIFF features from one frame into the next frame, the tracker outputs an affine transformation. We use this affine transformation to continuously update our estimate of any recognized object's location in the viewfinder. This in turn allows us to synchronize the movement of the augmentation layers with the movement of the objects, creating the illusion that the augmentations stick to the object. Note that an alternative motion classifier that uses the mobile device's accelerometer would be unable to estimate the affine transformation between frames at the same level of accuracy as the RIFF-based tracker and thus would be unable to precisely align the augmentation layers with the moving objects.

5.2 Hybrid Image Matching System

We now generalize the on-device image matching system of Section 5.1 into the hybrid image matching system depicted in Figure 5.10. This hybrid system can achieve a query expansion onto a remote server and an update of the local database simultaneously. First, at the beginning of every new query, the local database is searched. If a matching database candidate is found, as determined by a sufficient number of RANSAC inliers, then the search finishes locally on the device and no data is transmitted to the server. Occasionally, though, a good database match is not found on the device. At such a time, we transmit the REVV signatures of a group of query frames from the mobile device to the server. The compactness of REVV enables fast transmission even at a low uplink bitrate, while the discriminative capability of REVV ensures an accurate search through the remote database. When the server replies in the downlink with the labels, REVV signatures, and local features for the top-ranked database candidates, the mobile device can use the received feedback data to improve the results presented to the user on the current query and to update the

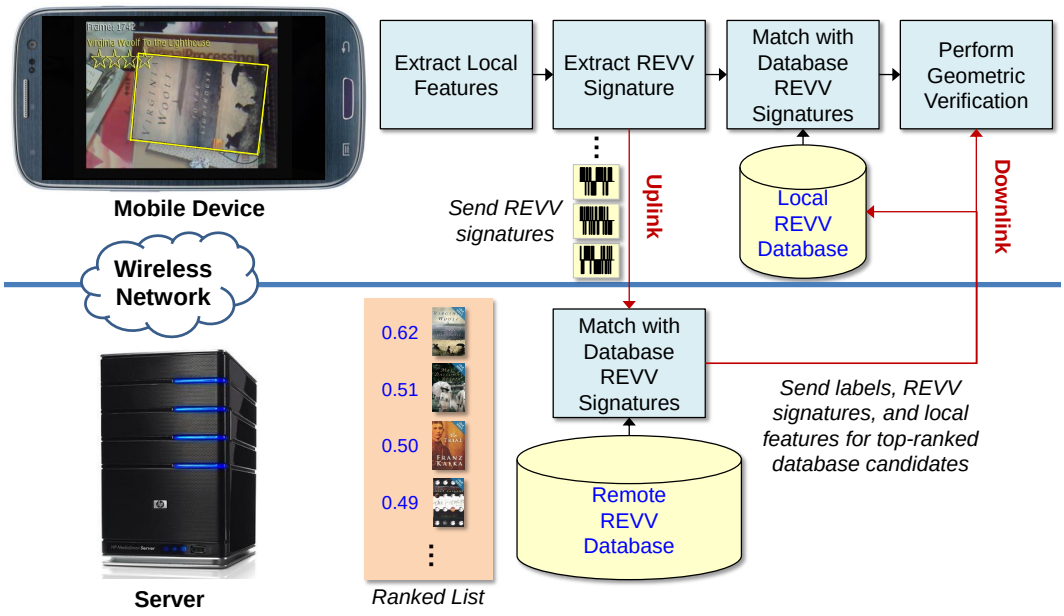


Figure 5.10: Hybrid image matching system. First, the local on-device database is searched. Then, if needed, the query is expanded onto a remote server. When the server replies, the local on-device database is updated.

local on-device database for future queries. This hybrid system is well suited to most wireless networks, where the uplink speed is much lower than the downlink speed.

Because we are processing viewfinder frames rather than still images, we would like to send a continuous stream of information from the mobile device to the server to achieve high responsiveness to sudden and important changes in the scene contents. The problem of compressing local features for a sequence of viewfinder frames has only recently been investigated [136, 137, 135]. The combination of a temporally coherent keypoint detector and interframe feature coding techniques presented in [136, 137, 135] enable a continuous stream of local features to be sent from the mobile device to the server at a low bitrate. The related problem of sending a continuous stream of global signatures like REVV has not been previously studied. Thus, in Section 5.2.1, we develop efficient methods for interframe coding of global signatures that can reduce the uplink bitrate by almost two orders of magnitude compared to independent coding of global signatures. This effectively paves the way for the hybrid system in Figure 5.10 to work well even for networks with low data transmission rates.

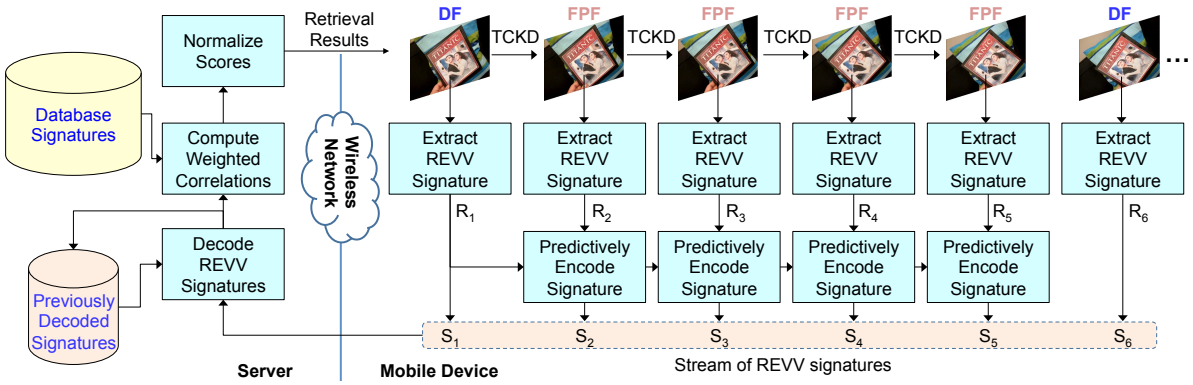


Figure 5.11: Interframe coding of REVV signatures extracted from a sequence of viewfinder frames and transmitted to a server for a remote database search.

5.2.1 Interframe Coding of Global Signatures

Our interframe coding framework is illustrated in Figure 5.11. First, to fully exploit the correlation between neighboring REVV signatures, we use a temporally coherent keypoint detector (TCKD) [136, 137, 135] that divides the acquired video frames into two categories: detection frames (DFs) and forward propagation frames (FPFs). For each DF, independent detection of SIFT keypoints is performed. Then, each SIFT keypoint is propagated into the subsequent FPF by searching across a small set of similarity transforms and minimizing the sum of absolute differences (SAD) between the SIFT keypoint's canonical patch in the DF and the transformed canonical patch in the FPF. This propagation continues until the next DF appears in the sequence, or until too many of the feature keypoints are not successfully propagated due to high SAD values. In Figure 5.11, we show the TCKD inserting a DF once for every 5 frames, but in practice, a DF is usually inserted once for every 30 frames.

To construct temporally coherent REVV signatures, we use the TCKD to detect SIFT keypoints for every viewfinder frame. Then, we extract SIFT descriptors from these keypoints and generate REVV signatures from the descriptors. Each DF's REVV signature is independently coded and transmitted, but each FPF's REVV signature is predictively coded. In the next three sections, we develop three different predictive coding methods that can adapt to various types of scene contents captured by the mobile device's camera and achieve the best overall coding efficiency.

Selective Codeword Propagation

For a codebook of k visual words, let the original REVV signature of frame t be denote as $\mathbf{R}_t = \left\{\left(u^R_{t,1}, \mathbf{b}^R_{t,1}\right), \cdots, \left(u^R_{t,k}, \mathbf{b}^R_{t,k}\right)\right\}$. Here, $u^R_{t,i} \in \{0,1\}$ is a binary variable indicating if the i^{th} codeword is visited by frame t, and $\mathbf{b}^R_{t,i} \in \{0,1\}^{l_{\text{lda}}}$ is the corresponding binary residual vector if the codeword is visited. Similarly, let the predictively coded REVV signature which is sent to the server be denoted as $\mathbf{S}_t = \left\{\left(u^S_{t,1}, \mathbf{b}^S_{t,1}\right), \cdots, \left(u^S_{t,k}, \mathbf{b}^S_{t,k}\right)\right\}$. For a DF, $\mathbf{S}_t = \mathbf{R}_t$, meaning the original REVV signature is transmitted. For an FPF, the selective codeword propagation (SCP) method assigns $u^S_{t,i} = \text{AND}\left(u^R_{t,i}, u^S_{t-1,i}\right)$ for $1 \leq i \leq k$. If $u^S_{t,i} = 1$, then the SCP method further assigns $\mathbf{b}^S_{t,i} = \mathbf{b}^S_{t-1,i}$, which propagates the previously sent binary residual vector for the i^{th} codeword. We do not encode the difference between the residual vectors $\mathbf{b}^R_{t,i}$ and $\mathbf{b}^S_{t-1,i}$, because these small differences are expensive to describe in terms of bitrate and are caused by small temporal fluctuations in the feature descriptors that do not noticeably affect retrieval results. Only $u^S_{t,i}$ needs to be sent for each FPF, because $\mathbf{b}^S_{t,i} = \mathbf{b}^S_{t-1,i}$ has been previously received at the server. Additionally, $u^S_{t,i}$ needs to be sent only when $u^S_{t-1,i} = 1$, because $u^S_{t,i} = 0$ when $u^S_{t-1,i} = 0$.

Selective Frame Propagation

When the scene content changes gradually, two consecutive frames visit mostly the same codewords and have similar binary residual vectors at these codewords. Taking the idea behind SCP encoding one step further, the selective frame propagation (SFP) method propagates all of the residual vectors between two frames if these two frames' REVV signatures have a high degree of similarity. As before, let the original REVV signature of frame t be denoted as $\mathbf{R}_t = \left\{\left(u^R_{t,1}, \mathbf{b}^R_{t,1}\right), \cdots, \left(u^R_{t,k}, \mathbf{b}^R_{t,k}\right)\right\}$ and the transmitted REVV signature be denoted as $\mathbf{S}_t = \left\{\left(u^S_{t,1}, \mathbf{b}^S_{t,1}\right), \cdots, \left(u^S_{t,k}, \mathbf{b}^S_{t,k}\right)\right\}$. We define the interframe codeword similarity between frame t and frame $t-1$ as follows:

$$r_k(t, t-1) = \frac{\sum_{i=1}^{k} \text{AND}\left(u^R_{t,i} = u^S_{t-1,i}\right)}{\sum_{i=1}^{k} u^R_{t,i}}. \tag{5.1}$$

If $r_k(t, t-1)$ exceeds a high threshold t_{r_k}, then SFP assigns $u^S_{t,i} = u^S_{t-1,i}$ and $\mathbf{b}^S_{t,i} = \mathbf{b}^S_{t-1,i}$ for $1 \leq i \leq k$. Only a single bit is sent to the server to indicate that the previous frame's REVV signature should be entirely propagated at every codeword. Otherwise, if $r_k(t, t-1)$ falls below t_{r_k}, then SFP switches back to SCP encoding; in this case, a single bit is sent to the server to indicate a temporary activation of the SCP mode, followed by the bits generated normally by SCP.

Selective Frame Propagation + Local Search

The selective frame propagation + local search (SFP + LS) method fully combines the advantages of query expansion onto a remote server and local search on the mobile device. If the local search results in a database match with a RANSAC inlier count N_{RANSAC} higher than a threshold $t_{\text{RANSAC}} = 25$ feature matches, then the search finishes locally on the mobile device. Otherwise, a REVV signature is transmitted to the server by SFP encoding. In this case, the first DF in the coding chain occurs on the first frame during which the local search fails. When the server replies with the labels, REVV signatures, and local features of the top candidates, the local on-device database is opportunistically updated, so that (i) the current query can be improved by selecting from the best database candidates both locally and remotely, and (ii) future local searches are more likely to succeed with the updated local database.

5.2.2 Analysis of Coding Performance

The retrieval accuracy of the SCP and SFP methods of Section 5.2.1 can be predicted with the statistical model developed for REVV in Section 4.3. Let the retrieval precision predicted by the model for these two methods be denoted as $\text{PA1}_{\text{remote}}$. Because the SFP + LS method uses a hybrid combination of local database search and remote database search, the overall retrieval precision is $\text{PA1}_{\text{hybrid}} = p_{\text{local}} + (1 - p_{\text{local}})\,\text{PA1}_{\text{remote}}$, where p_{local} is the probability that the local search succeeds. In general, p_{local} will slowly increase over time as the local database is updated.

The uplink bitrate for the various coding methods can also be accurately modeled.

For independent coding of REVV signatures, the uplink bitrate (in bits/second) is

$$R_{\text{Indep}} = N_{\text{Frames}}\, k\, (1 + p_{\text{visit}}\, l_{\text{lda}}) \tag{5.2}$$

where N_{Frames} is the number of viewfinder frames per second, p_{visit} is the probability that a codeword is visited by an image, and l_{lda} is the residual vector dimensionality after cell-specific LDA. For SCP, the uplink bitrate (in bits/second) is

$$R_{\text{SCP}} = \underbrace{N_{\text{DF}}\, k\, (1 + p_{\text{visit}}\, l_{\text{lda}})}_{\text{bitrate for DFs}} + \underbrace{N_{\text{FPF}}\, k\, p_{\text{visit}}}_{\text{bitrate for FPFs}} \tag{5.3}$$

where N_{DF} and N_{FPF} are the number of DFs and FPFs, respectively, per second. Note that $N_{\text{Frames}} = N_{\text{DF}} + N_{\text{FPF}}$. Similarly, the uplink bitrate (in bits/second) for SFP is

$$R_{\text{SFP}} = \underbrace{N_{\text{DF}}\, k\, (1 + p_{\text{visit}}\, l_{\text{lda}})}_{\text{bitrate for DFs}} + \underbrace{N_{\text{FPF}}\, (1 + k\, p_{\text{visit}}\, P\, (r_k < t_{r_k}))}_{\text{bitrate for FPFs}} \tag{5.4}$$

where $P\,(r_k < t_{r_k})$ is that probability that the interframe codeword similarity r_k falls below the threshold t_{r_k}. Finally, the uplink bitrate (in bits/second) for SFP + LS is

$$R_{\text{SFP+LS}} = (1 - p_{\text{local}})\, R_{\text{SFP}} \tag{5.5}$$

since the uplink bitrate is 0 when local search succeeds.

The savings between independent coding and SCP coding can be expressed as

$$\begin{aligned} \Delta R_{\text{SCP}} &= R_{\text{Indep}} - R_{\text{SCP}} && (5.6)\\ &= N_{\text{FPF}}\, k\, [1 + p_{\text{visit}}\, (l_{\text{lda}} - 1)]\,. && (5.7) \end{aligned}$$

Thus, the bitrate savings gained by SCP increase as the number of FPFs per second increases. Similarly, the savings between independent coding and SFP coding can be written as

$$\begin{aligned} \Delta R_{\text{SFP}} &= R_{\text{Indep}} - R_{\text{SFP}} && (5.8)\\ &= \Delta R_{\text{SCP}} + N_{\text{FPF}}\, [k\, p_{\text{visit}}\, P\, (r_k \geq t_{r_k}) - 1]\,. && (5.9) \end{aligned}$$

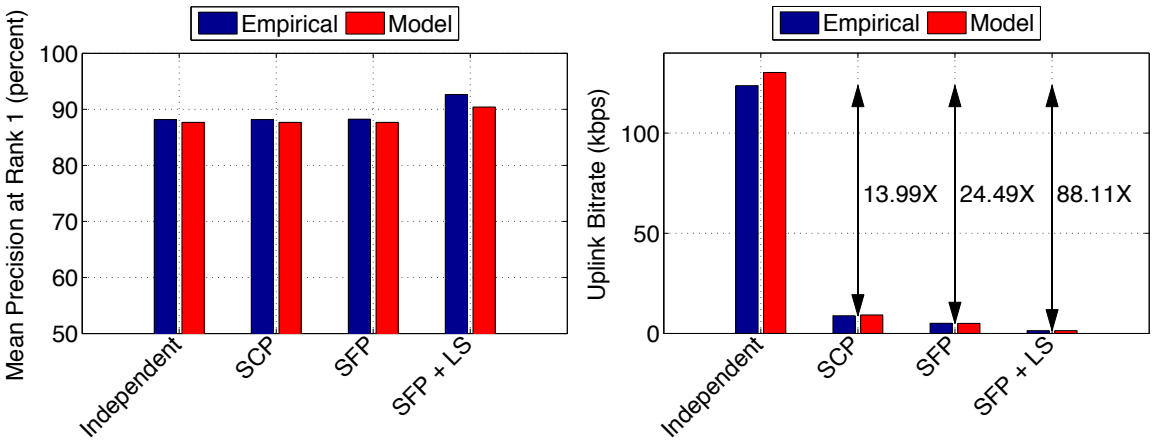


Figure 5.12: (a) Mean precision at rank 1 (PA1) and (b) Uplink bitrate (in kbps) for four different coding methods on the SSMAR Dataset.

We see that the bitrate savings offered by SFP are greater than the bitrate savings offered by SCP. When the interframe codeword similarity r_k increases, the probability $P\left(r_k \geq t_{r_k}\right)$ increases and leads to larger bitrate reductions by SFP on top of the bitrate savings already provided by SCP. Lastly, the savings between independent coding and SFP + LS can be written as

$$\begin{aligned} \Delta R_{\text{SFP+LS}} &= R_{\text{Indep}} - R_{\text{SFP+LS}} && (5.10) \\ &= \Delta R_{\text{SFP}} + p_{\text{local}}\, R_{\text{SFP}}. && (5.11) \end{aligned}$$

Hence, the bitrate savings gained by SFP + LS are even larger than the bitrate savings gained by SFP. The additional savings in the amount of $p_{\text{local}}\, R_{\text{SFP}}$ increases as the probability of success in local search p_{local} increases.

5.2.3 Interframe Coding Results

To test the effectiveness of our interframe coding methods, we perform an evaluation on the Stanford Streaming Mobile Augmented Reality (SSMAR) Dataset described in Appendix B. There are 32 query videos and a database of 1M still images. For every query DF, we extract $N_{\text{feat}} = 250$ SIFT features using the same feature selector

[76] that we used in Chapters 3 and 4. For REVV, we use a codebook of $k = 190$ codewords, $l_{\text{pca}} = 64$ global PCA eigenvectors, and $l_{\text{lda}} = 32$ cell-specific LDA eigenvectors. For interframe coding, we use only 1 DF for every 30 frames and an interframe codeword similarity threshold of $t_{r_k} = 0.9$.

First, Figure 5.12(a) plots the mean precision at rank 1 (PA1) measurements averaged across the 32 different query videos contained in the SSMAR Dataset. The PA1 values are measured separately for independent coding, SCP, SFP, and SFP + LS. It can be seen that SCP and SFP perform vary similar to independent coding, showing that SCP and SFP can effectively adapt to changes in the videos and preserve a high retrieval accuracy. The fourth method, SFP + LS, attains a slightly higher retrieval accuracy, because geometric verification is performed after comparing REVV signatures and further removes some false database candidates. Figure 5.12(a) plots the empirical PA1 values next to the predicted PA1 values derived from our statistical model. The model closely predicts the retrieval accuracy of our coding methods.

Then, Figure 5.12(b) plots the uplink bitrate (in kbps) for the same four coding methods. Once again, our statistical model accurately predicts the bitrate required for each coding method. Independent coding requires around 120 kbps. With interframe coding, we can substantially reduce the uplink bitrate required to communicate the REVV signatures: by a factor of 14× for SCP, 24× for SFP, and 88× for SFP + LS. The bitrate reduction achieved by the best performing method, SFP + LS, is a combined result of two key factors: (i) the temporal redundancy between REVV signatures of viewfinder frames is carefully exploited, and (ii) when the local database search is sufficient, a query does not need to be sent to the remote server. The downlink bitrate for the independent coding, SCP, and SFP methods is close to 0 kbps, because the server replies with just the label and metadata for the top ranked database candidates. The downlink bitrate for SFP + LS is 14 kbps, where the server replies with the labels, metadata, REVV signature, and a small set of compressed local features for the top ranked database candidates. The low ratio of uplink bitrate to downlink rate for SFP + LS is well suited to typical wireless networks that have much lower uplink speeds compared to downlink speeds.

5.3 Summary

In this chapter, we have developed practical MVS systems that can achieve low latencies and high retrieval accuracies. These systems use the memory-efficient image databases that we created previously. First, we studied how to build an on-device image matching system that runs entirely on a mobile device. With REVV signatures, we can store a database of 100K images in about 50 MB of RAM, which is available on almost 100 percent of mobile devices today. Since this system performs fast local database comparisons with REVV signatures, a very low recognition latency can be guaranteed, regardless of external network or server conditions. Achieving a consistently low recognition latency is required to generate continuous augmentations in the viewfinder for applications such as media cover recognition, book spine recognition, and outdoor landmark recognition. Our system also uses a motion-adaptive query selection mechanism to automatically infer the user's interest, so the user never has to press a button to manually start a query and can expect to see new results appearing almost instantly in the viewfinder. As an added advantage, motion-adaptive query selection picks high-quality query frames which are not degraded by motion blur, thereby increasing the probability of a successful query result and reducing the probability of wasting battery power over processing blurry query frames.

Second, we researched how the local database can be opportunistically updated, by occasionally performing a query expansion onto the remote server. This results in a hybrid system that combines the advantages of both local and remote database searching. Most of the time, the local search is sufficient and no data is sent to the remote server. When we do need to contact the remote server, we transmit a stream of REVV signatures extracted from the viewfinder frames to search the database stored on the server. By developing interframe REVV coding methods, we can reduce the uplink bitrate by almost two orders of magnitude compared to independent coding of REVV signatures. Less than 2 kbps is required for the uplink bitrate, making this query expansion and database update method feasible even in networks with low uplink speeds. Searching the remote database improves the accuracy of the current query and expands the coverage of the local database for future queries.

Chapter 6

Conclusions

6.1 Lessons Learned

A memory-efficient image database can substantially improve the overall efficiency and reduce the total latency in different mobile visual search (MVS) systems. A compact database representation enables us to store more images in a database, either on a server or on a mobile device, which expands the range of objects that can be effectively searched with an MVS system. Effective database compression opens the possibility of new on-device image matching and hybrid image matching architectures that would otherwise not be possible.

First, we studied how to reduce the memory usage of databases built from feature histograms, including the popular vocabulary tree framework. Without compression, feature histograms for a large database (e.g., 1M images) consume a large amount of memory, which can easily lead to performance bottlenecks on a memory-congested server or prevent their storage on a mobile device with a small memory capacity. However, we have found that there is a significant amount of statistical redundancy and extraneous information in the feature histograms that can be eliminated without hurting the retrieval accuracy. We developed one technique called tree histogram coding (THC) to directly compress the tree histograms and a second technique called inverted index coding (IIC) to compress an inverted index constructed from database tree histograms. Both techniques can reduce the memory usage by a factor of $4-$

5×, with no adverse effects on the retrieval accuracy. In developing THC and IIC, we utilized word-aligned codecs that respect computer word boundaries in storing codewords, leading to very low decoding delays: less than 20 ms per query for a database of 1M images. We also provided a statistical analysis of the memory savings achieved by THC and IIC, where our analysis showed that a large part of the savings are due to discarding the ordering information amongst a set of symbols.

Then, we studied how to create even more memory-efficient databases from feature residuals, culminating in the creation of the residual enhanced visual vector (REVV) as a compact, discriminative global image signature. Whereas feature histograms require a large codebook to achieve high retrieval accuracies, REVV can achieve the same retrieval accuracies while using a much smaller codebook. The REVV signatures not only reduce the database memory usage by $12 - 14\times$, but also enable direct comparisons in the compressed domain. REVV signatures for database images are independently encoded, so insertions, deletions, and updates of randomly selected database signatures can be easily performed. We developed a fast multi-round scoring algorithm so that a majority of irrelevant REVV signatures in the database are not searched during a query. We also performed a statistical analysis of REVV's retrieval performance, in order to optimize the number of codewords and the number of local features per image for a given memory budget.

Finally, we explored practical applications of our memory-efficient image databases for building low-latency MVS systems. A compact database is especially useful for an on-device image matching architecture, where the entire image matching pipeline is implemented on the mobile device and there is then no reliance on an external network or server for support. When the database is searched locally, we can ensure the MVS application can achieve a low retrieval latency anywhere and anytime. This is a very attractive property for mobile augmented reality (MAR) applications that continuously require low recognition delays. Our on-device image matching system is able to recognize objects like media covers, book spines, and outdoor landmarks in under 1 second out of a database of 100K images, and our system performs continuous augmentation for the recognized objects in the viewfinder by a combination of on-device visual tracking and motion-adaptive query selection. To update the local

database, we perform a query expansion by sending a compact stream of interframe-coded REVV signatures from the mobile device to the server, and we update the local on-device database with the information sent back by the server. Although REVV signatures are already compact, effective interframe coding can further reduce the uplink bitrate of sending a continuous stream of REVV signatures by nearly two orders of magnitude, to less than 2 kbps.

6.2 Future Work

A global image signature, such as a tree histogram or a REVV signature, works well at filtering away the majority of irrelevant database images, while retaining the relevant database images in a shortlist of top candidates. This shortlist can be further processed through a geometric verification stage that considers the spatial consistency of local feature matches. A future global signature can perform both fast ranking based on descriptors in the first stage and more complex geometric comparisons based on geometric information in the second stage. An interesting research question is: How can the descriptors and geometric information of an image be jointly summarized and organized into a discriminative yet compact global signature?

Global image signatures have been developed based on local gradient descriptors in this thesis. Often, for glossy or texture-limited surfaces, the shape and color of an object can be more discriminative cues than local intensity gradients. It will be interesting and useful to study the development of global image signatures based on other types of image features such as shape or color descriptors. Does forming feature histograms or feature residuals over sets of shape or color descriptors lead to effective retrieval performance, or is a different retrieval method required to effectively index these other types of descriptors? Also, how can global signatures generated from shape and color descriptors be combined with global signatures generated from local gradient descriptors to achieve the best overall object recognition accuracy?

Features like SIFT or SURF also perform poorly for recognizing documents containing dense text, where the repetitive structure of the text characters can cause

confusion for feature-based image matching. Text-aware descriptors have been developed with much higher recognition accuracy [209]. Developing a compact global signature for these text-aware descriptors can aid the development of memory-efficient image databases that index large text document collections, e.g., all of the papers and books in a particular subject or field.

There also exist more innovative applications of compact image databases that can be explored. With the emerging wave of camera-enabled smart glasses and wearable devices, storing a local image database on these devices can provide fast recognition capabilities anywhere the user travels. A smart automobile which has an array of cameras recording its surroundings is another ideal platform for visual search. A local on-device database can aid the detection and recognition of outdoor landmarks, road signs, lane markings, pedestrians, and other vehicles. Ideally, smart wearable devices and smart automobiles would also be able to communicate with one another in a collaborative framework, where they can share and enhance each other's visual recognition results

Appendix A

MPEG CDVS Dataset

When a database is hosted on a remote server, the query signatures extracted from images taken by a mobile device must be transmitted over a wireless network to be compared to image signatures stored in this database. It is critically important that the amount of data sent over the network is as small as possible to minimize the overall system latency in mobile visual search (MVS) systems. The Motion Picture Experts Group (MPEG) have developed an emerging standard on Compact Descriptors for Visual Search (CDVS) that aims to design a low bitrate signature for general image matching applications [80, 176]. To evaluate the performance of different image signatures, the MPEG CDVS Dataset has been developed. The dataset consists of five categories of images: graphics, paintings, video frames, landmarks, and common objects. These images are selected from a combination of the Stanford Mobile Visual Search Dataset [32], the Zurich Buildings Dataset [190], the Peking University Buildings Dataset [123], the Telecom Italia Buildings Dataset [75], and the University of Kentucky Benchmark Dataset [198]. Table A.1 lists the number of query images and database images contained in each of these five categories. Then, Figure A.1 shows some examples of matching query and database images. The dataset contains a heterogeneous assortment of images and a variety of challenging image distortions, so a performance evaluation over this dataset gives a good indication of how well algorithms perform in practical MVS systems.

Category	Query Images	Database Images
Graphics	1,500	1,000
Paintings	364	91
Video Frames	400	100
Landmarks	3,499	9,559
Common Objects	2,550	7,650
All Categories	8,313	18,440

Table A.1: Number of query images and database images in the five categories of the MPEG CDVS Dataset.

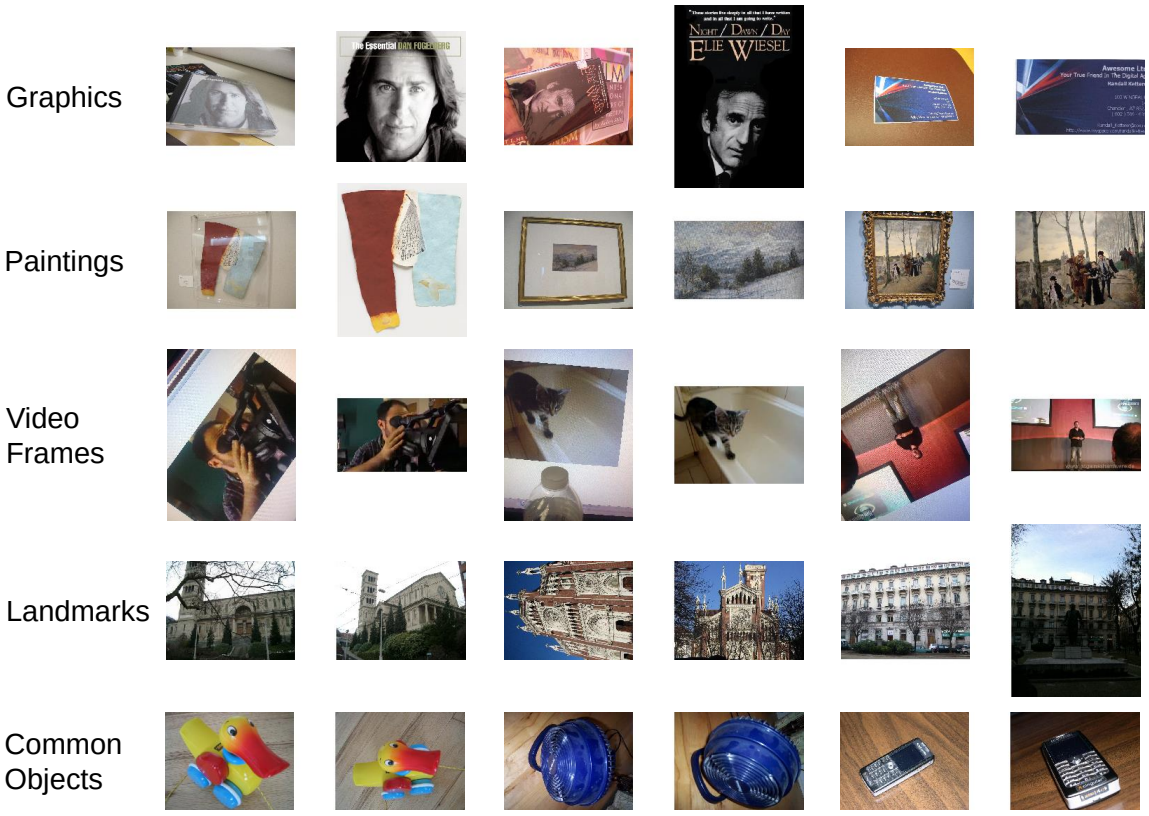


Figure A.1: Examples for pairs of matching query images (left of each pair) and database images (right of each pair) from the MPEG CDVS Dataset. Three different pairs are shown for each of the five image categories.

Since this thesis focuses on large-scale image retrieval, we are particularly interested in how accurately we can match query images in these five categories against a

large set of database images. The 18,440 database images of the five aforementioned categories are embedded into a much larger set of 1M distractor images to assess large-scale retrieval performance. The retrieval accuracy is measured in terms of the mean precision at rank 1 (PA1) and the mean average precision (MAP) values. First, we define the precision at a given cut-off rank r for the q^{th} query image as

$$P(q,r) = \frac{\sum_{r'=1}^{r} \text{rel}(q,r)}{r} \tag{A.1}$$

where $\text{rel}(q,r) = 1$ if the database image at rank r in the ranked list for the q^{th} query image is relevant to the q^{th} query image, and $\text{rel}(q,r) = 0$ otherwise. Hence, the numerator of Equation A.1 is the total number of relevant database images for the q^{th} query image, when the ranked list contains r database candidates. Next, we define the average precision for the q^{th} query image as

$$AP(q,N) = \frac{1}{R(q)} \sum_{r=1}^{N} P(q,r)\,\text{rel}(q,r) \tag{A.2}$$

where N is the maximal size of the ranked list, and $R(q)$ is the number of relevant database images for the q^{th} query image. Then, the PA1 value for a set of Q query images is

$$\text{PA1} = \frac{1}{Q} \sum_{q=1}^{Q} P(q,1) \tag{A.3}$$

which is equal to the probability that the top-ranked database candidate is relevant. Similarly, the MAP value for the same set of Q query images can be defined as

$$\text{MAP} = \frac{1}{Q} \sum_{q=1}^{Q} AP(q,N). \tag{A.4}$$

For the MPEG CDVS Dataset, the CDVS standard sets a maximal ranked list size of $N = 500$. The MAP value assesses how far down the ranked list the relevant database images are ranked by the retrieval algorithm. In general, the PA1 and MAP values are highly correlated for all of the algorithms we have tested in this thesis.

Appendix B

SSMAR Dataset

In Chapter 5, we extract a continuous stream of REVV signatures from viewfinder frames as part of a hybrid image matching system. In order to effectively compress this stream, we develop several interframe coding methods for the REVV signature. Our evaluation of the effectiveness of these interframe coding methods utilizes the publicly available Stanford Streaming Mobile Augmented Reality (SSMAR) Dataset developed by Makar et al. [136, 137]. This dataset contains 32 different VGA-resolution videos recorded with a camera-phone at a frame rate of 30 frames/second. There are substantial amounts of camera motion (pan, tilt, zoom), glare and reflections, motion blur, and background clutter in each query video. The dataset also contains 23 labeled database objects spanning four different categories: books, CD covers, DVD covers, and common household objects. Figure B.1 shows some examples of matching query frames and database images. Figure B.2 shows keyframes sampled twice per second from one of the query videos. To assess large-scale retrieval accuracy with this dataset, we embed the 23 labeled database images into the same set of 1M distractor images that was used for the MPEG CDVS Dataset described in Appendix A. Retrieval precision measures like mean precision at rank 1 (PA1) and mean average precision (MAP) are also identical to those defined in Appendix A.

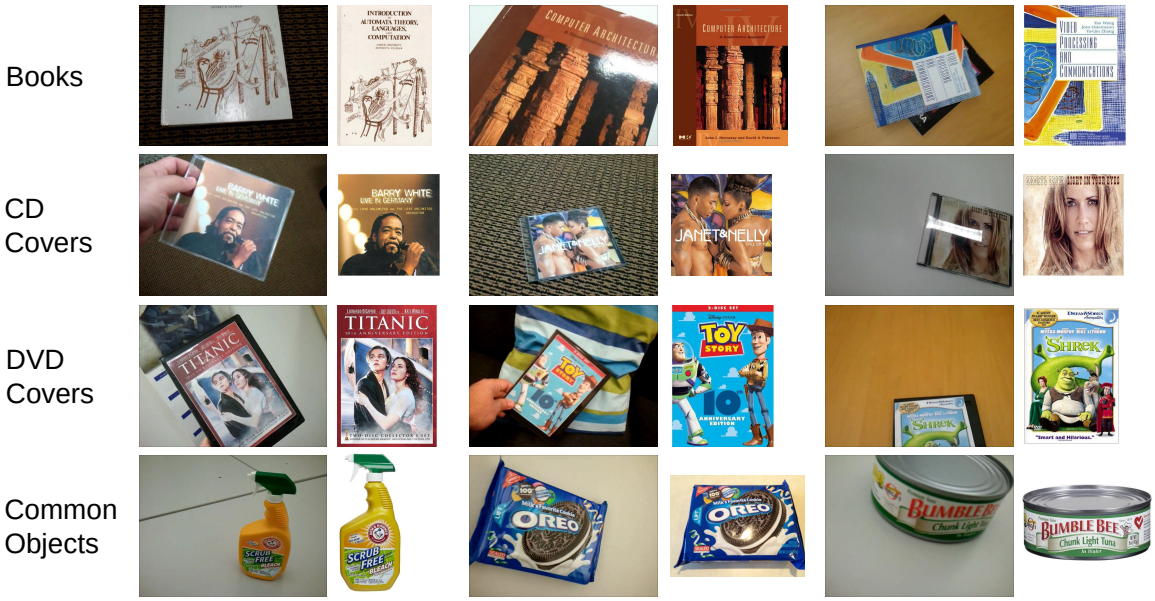


Figure B.1: Examples for pairs of matching query frames (left of each pair) and database images (right of each pair) from the SSMAR Dataset. Three different pairs are shown for each of the four image categories.

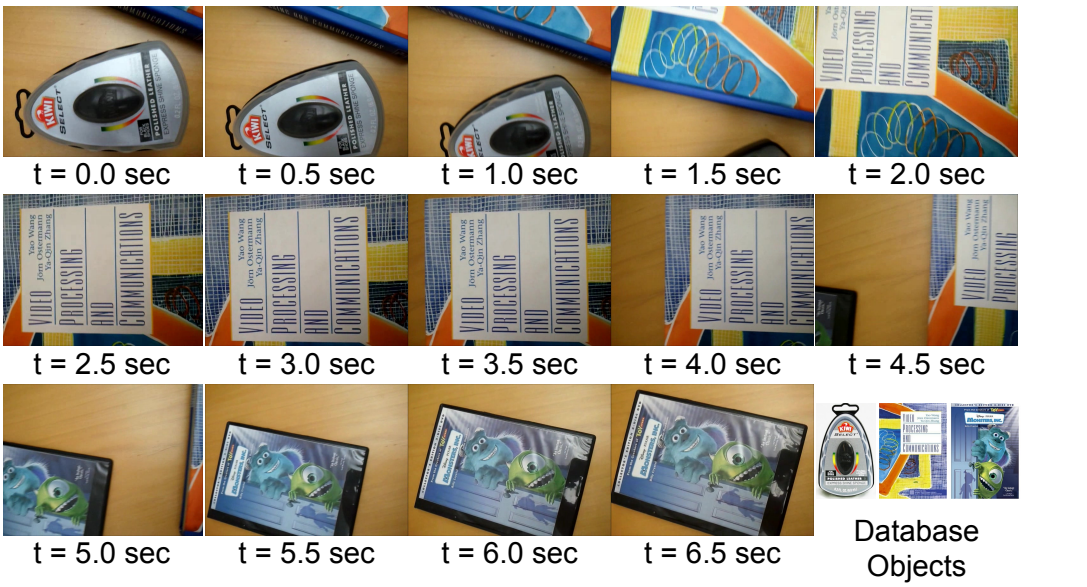


Figure B.2: Keyframes which are sampled twice per second from one of the query videos in the SSMAR Dataset.

Appendix C

Training Dataset

Throughout this thesis, a training dataset used for algorithm developement is completely separated from the testing datasets such as the MPEG CDVS Dataset described in Appendix A or the SSMAR Dataset described in Appendix B. This training dataset is used for (i) building vocabulary trees and estimating various probability distributions for feature histograms in Chapter 3, and (ii) generating codebooks, eigenvectors, and correlation weights for feature residuals in Chapter 4. This ensures the training data is never mixed with the testing data in any of our experiments.

The training dataset combines images from the Oxford Buildings Dataset [174], the Caltech Buildings Dataset [5], and the INRIA Holidays Dataset [101] to build a set of matching image pairs and a set of non-matching image pairs. Table C.1 lists the number of image pairs selected from each of the three external datasets. In total, 11,570 matching and 11,570 non-matching image pairs are available in the training set. Figure C.1 shows a few examples of these image pairs. It can be observed that

Dataset	**Matching Pairs**	**Non-matching Pairs**
Oxford Buildings Dataset [174]	7,695	7,695
Caltech Buildings Dataset [5]	1,875	1,875
INRIA Holidays Dataset [101]	2,072	2,072
Combined	11,570	11,570

Table C.1: Number of matching image pairs and non-matching image pairs selected from various external datasets for inclusion in the training dataset.

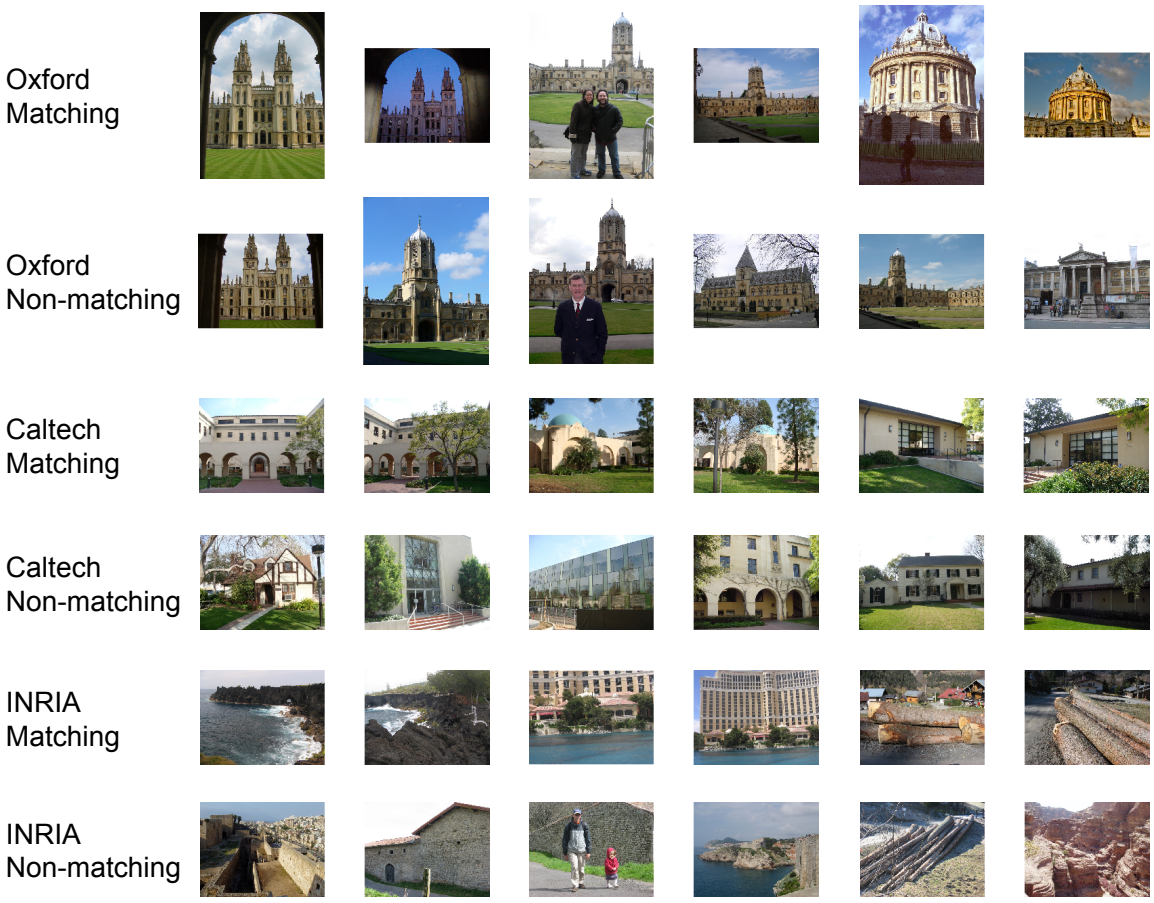


Figure C.1: Examples of matching image pairs and non-matching image pairs from the Oxford Buildings Dataset [174], the Caltech Buildings Dataset [5], and the INRIA Holidays Dataset [101].

significant geometric and photometric distortions can separate the matching images. At the same time, similar visual elements often exist between non-matching images, such as similar architectural elements on different buildings or similar textures on different walls. To find matching feature descriptors between a pair of matching images, we utilize a combination of the distance ratio test [134] and RANSAC with an affine model [71].

The training dataset also contains a set of 1M images, which are completely independent of the set of 1M distractor images used in the MPEG CDVS Dataset. The 1M training images are downloaded from Flickr and Panoramio and include photos

of landmarks, natural scenery, events, and people. By extracting a few hundred local features from each of the 1M training images, we can accumulate enough local features to accurately train any codebook, including the largest vocabulary trees that we have constructed in Chapter 3.

Bibliography

[1] A. Adams, N. Gelfand, and K. Pulli. Viewfinder alignment. *Computer Graphics Forum*, 27(2):597–606, April 2008.

[2] A. Adams, E.-V. Talvala, S. H. Park, D. E. Jacobs, B. Ajdin, N. Gelfand, J. Dolson, D. Vaquero, J. Baek, M. Tico, H. P. A. Lensch, W. Matusik, K. Pulli, M. Horowitz, and M. Levoy. The Frankencamera: an experimental platform for computational photography. In *ACM SIGGRAPH*, July 2010.

[3] A. Alahi, R. Ortiz, and P. Vandergheynst. FREAK: Fast retina keypoint. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 510–517, June 2012.

[4] P. F. Alcantarilla, A. Bartoli, and A. J. Davison. KAZE features. In *European Conference on Computer Vision*, pages 214–227, October 2012.

[5] M. Aly, P. Welinder, M. Munich, and P. Perona. *Caltech Buildings Dataset*, June 2009. http://tinyurl.com/n2p8jbk.

[6] Amazon. Amazon Flow, 2013. http://www.amazon.com/A9-Innovations-LLC-Powered-Amazon/dp/B008G318PE.

[7] M. Ambai and Y. Yoshida. CARD: Compact and real-time descriptors. In *IEEE International Conference on Computer Vision*, pages 97–104, November 2011.

[8] V. N. Anh and A. Moffat. Inverted index compression using word-aligned binary codes. *Information Retrieval*, 8(1):151–166, January 2005.

[9] V. N. Anh and A. Moffat. Improved word-aligned binary compression for text indexing. *IEEE Transactions on Knowledge and Data Engineering*, 18(6):857–861, June 2006.

[10] R. Arandjelovic. Three things everyone should know to improve object retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2911–2918, June 2012.

[11] R. Arandjelovic and A. Zisserman. All about VLAD. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1578–1585, June 2013.

[12] S. Arya and D. M. Mount. Algorithms for fast vector quantization. In *IEEE Data Compression Conference*, pages 381–390, April 1993.

[13] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, November 1998.

[14] G. Baatz, K. Koser, D. Chen, R. Grzeszczuk, and M. Pollefeys. Leveraging 3d city models for rotation invariant place-of-interest recognition. *International Journal of Computer Vision*, 96(3):315–334, February 2012.

[15] A. Babenko and V. Lempitsky. The inverted multi-index. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3069–3076, June 2012.

[16] S. K. Bar-Lev and P. Enis. On the classical choice of variance stabilizing transformations and an application for a Poisson variate. *Biometrika*, 75(4):803–804, 1988.

[17] R. Basri and D. W. Jacobs. Recognition using region correspondences. *International Journal of Computer Vision*, 25(2):145–166, November 1997.

[18] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, June 2008.

[19] H. Bay, B. Fasel, and L. V. Gool. Interactive museum guide: Fast and robust recognition of museum objects. In *IEEE International Workshop on Mobile Vision*, pages 179–191, May 2006.

[20] H. Bay, T. Tuytelaars, and L. V. Gool. SURF: Speeded up robust features. In *European Conference on Computer Vision*, pages 404–417, May 2006.

[21] J. L. Bentley. K-d trees for semidynamic point sets. In *Symposium on Computational Geometry*, pages 187–197, June 1990.

[22] D. Blandford and G. Blelloch. Index compression through document reordering. In *IEEE Data Compression Conference*, page 342, March 2002.

[23] A. Bookstein, S. T. Klein, and T. Raita. Markov models for clusters in concordance compression. In *IEEE Data Compression Conference*, pages 116–125, March 1994.

[24] A. Bookstein, P. Lankinen, and C. W. Sze. Modeling word occurrences for the compression of concordances. In *IEEE Data Compression Conference*, pages 254–290, March 1995.

[25] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, NY, USA, 1984.

[26] M. Brown, G. Hua, and S. Winder. Discriminative learning of local image descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):43–57, January 2011.

[27] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua. BRIEF: Computing a Local Binary Descriptor Very Fast. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1281–1298, July 2012.

[28] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary robust independent elementary features. In *European Conference on Computer Vision*, pages 778–792, September 2010.

[29] Y. Cao, C. Wang, Z. Li, L. Zhang, and L. Zhang. Spatial-bag-of-features. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3352–3359, June 2010.

[30] V. Chandrasekhar. *Low-Bitrate Image Retrieval with Compressed Histogram of Gradients Descriptors*. PhD thesis, Stanford University, February 2013.

[31] V. Chandrasekhar, D. Chen, Z. Li, G. Takacs, S. Tsai, R. Grzeszczuk, and B. Girod. Low-rate image retrieval with tree histogram coding. In *International Conference on Mobile Multimedia Communications*, pages 7:1–7:7, September 2009.

[32] V. Chandrasekhar, D. Chen, S. Tsai, N.-M. Cheung, H. Chen, G. Takacs, Y. Reznik, R. Vedantham, R. Grzeszczuk, J. Bach, and B. Girod. The Stanford mobile visual search data set. In *ACM Conference on Multimedia Systems*, pages 117–122, February 2011.

[33] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, R. Grzeszczuk, and B. Girod. CHoG: compressed histogram of gradients – a low bitrate feature descriptor. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2504–2511, June 2009.

[34] V. Chandrasekhar, G. Takacs, D. Chen, S. Tsai, Y. Reznik, R. Grzeszczuk, and B. Girod. Compressed histogram of gradients: a low bitrate descriptor. *International Journal of Computer Vision*, 96(3):384–399, 2012.

[35] D. Chen, G. Baatz, K. Koser, S. Tsai, R. Vedantham, T. Pylvanainen, K. Roimela, Xin Chen, J. Bach, M. Pollefeys, B. Girod, and R. Grzeszczuk. City-scale landmark identification on mobile devices. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 737–744, June 2011.

[36] D. Chen and B. Girod. Memory-efficient image databases for mobile visual search. *IEEE MultiMedia*, 21(1):14–23, January–March 2014.

[37] D. Chen, S. Tsai, V. Chandrasekhar, G. Takacs, H. Chen, R. Vedantham, R. Grzeszczuk, and B. Girod. Residual enhanced visual vectors for on-device image matching. In *IEEE Asilomar Conference on Signals, Systems, and Computers*, pages 850–854, November 2011.

[38] D. Chen, S. Tsai, V. Chandrasekhar, G. Takacs, J. P. Singh, and B. Girod. Tree histogram coding for mobile image matching. In *IEEE Data Compression Conference*, pages 143–152, March 2009.

[39] D. Chen, S. Tsai, V. Chandrasekhar, G. Takacs, R. Vedantham, R. Grzeszczuk, and B. Girod. Inverted index compression for scalable image matching. In *IEEE Data Compression Conference*, March 2010.

[40] D. Chen, S. Tsai, V. Chandrasekhar, G. Takacs, R. Vedantham, R. Grzeszczuk, and B. Girod. Residual enhanced visual vector as a compact signature for mobile visual search. *Signal Processing*, 93(8):2316–2327, August 2013.

[41] D. Chen, S. Tsai, C.-H. Hsu, K.-H. Kim, J. P. Singh, and B. Girod. Building book inventories using smartphones. In *ACM International Conference on Multimedia*, pages 651–654, October 2010.

[42] D. Chen, S. Tsai, C.-H. Hsu, J. P. Singh, and B. Girod. Mobile augmented reality for books on a shelf. In *IEEE Workshop on Visual Content Identification and Search*, pages 1–6, July 2011.

[43] D. Chen, S. Tsai, K.-H. Kim, C.-H. Hsu, J. P. Singh, and B. Girod. Low-cost asset tracking using location-aware camera phones. In *SPIE Conference on Applications of Digital Image Processing*, pages 77980R–77980R–13, August 2010.

[44] D. Chen, S. Tsai, R. Vedantham, R. Grzeszczuk, and B. Girod. Streaming mobile augmented reality on mobile phones. In *International Symposium on Mixed and Augmented Reality*, pages 181–182, October 2009.

[45] J. Choi and G. G. Medioni. StaRSaC: Stable random sample consensus for parameter estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 675–682, June 2009.

[46] P. A. Chou, T. D. Lookabaugh, and R. M. Gray. Optimal pruning with applications to tree-structured source coding and modeling. *IEEE Transactions on Information Theory*, 35(2):299–315, March 1989.

[47] P.A. Chou, T. Lookabaugh, and R. M. Gray. Entropy-constrained vector quantization. *IEEE Transactions on Acoustics, Speech and Signal*, 37(1):31–42, January 1989.

[48] O. Chum and J. Matas. Randomized RANSAC with T-d,d test. In *Image and Vision Computing*, volume 22, pages 837–842, February 2002.

[49] O. Chum and J. Matas. Matching with PROSAC: progressive sample consensus. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 220–226, June 2005.

[50] O. Chum and J. Matas. Geometric hashing with local affine frames. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 879–884, June 2006.

[51] O. Chum and J. Matas. Optimal randomized RANSAC. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(8):1472–1482, August 2008.

[52] O. Chum and J. Matas. Fast computation of min-hash signatures for image collections. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3077–3084, June 2012.

[53] O. Chum, A. Mikulik, M. Perdoch, and J. Matas. Total recall II: Query expansion revisited. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 889–896, June 2011.

[54] O. Chum, M. Perdoch, and J. Matas. Geometric min-hashing: Finding a (thick) needle in a haystack. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 17–24, June 2009.

[55] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *IEEE International Conference on Computer Vision*, pages 1–8, October 2007.

[56] O. Chum, J. Philbin, and A. Zisserman. Near duplicate image detection: min-hash and tf-idf weighting. In *British Machine Vision Conference*, pages 50.1–50.10, September 2008.

[57] O. Chum, T. Werner, and J. Matas. Epipolar geometry estimation via RANSAC: Benefits from the oriented epipolar constraint. In *International Conference on Pattern Recognition*, pages 112–115, August 2004.

[58] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *International Conference on Very Large Databases*, pages 426–435, August 1997.

[59] R. G. Cinbis. Image categorization using Fisher kernels of non-i.i.d. image models. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2184–2191, June 2012.

[60] D. Comer. Ubiquitous B-tree. *ACM Computing Surveys*, 11(2):121–137, June 1979.

[61] P. C. Cosman, K. Oehler, A. A. Heaton, and R. M. Gray. Tree-structured vector quantization with input-weighted distortion measures, November 1991.

[62] G. Cuellar, D. Eckles, and M. Spasojevic. Photos for information: a field study of cameraphone computer vision interactions in tourism. In *Conference on Human Factors in Computing Systems*, pages 3243–3248, April 2008.

[63] S. Deffeyes. Mobile augmented reality in the data center. *IBM Journal of Research and Development*, 55(5):5:1–5:5, 2011.

[64] M. Douze, H. Jegou, C. Schmid, and P. Perez. Compact video description for copy detection with precise temporal alignment. In *European Conference on Computer Vision*, pages 522–535, September 2010.

[65] M. Douze, A. Ramisa, and C. Schmid. Combining attributes and Fisher vectors for efficient image retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 745–752, June 2011.

[66] Z. Drezner and N. Farnum. A generalized Binomial distribution. *Communications in Statistics*, 22(11):3051–3063, 1993.

[67] Epson. Epson Moverio BT-100 wearable display. http://www.epson.com/cgi-bin/Store/jsp/Moverio/Home.do.

[68] A. Farano. Quadmented, 2011. http://adrianofarano.com/2011/02/quadmented-combines-ar-and-storytelling.

[69] N. Farvardin and J. W. Modestino. Optimum quantizer performance for a class of non-Gaussian memoryless sources. *IEEE Transactions on Information Theory*, 30(3):485–497, May 1984.

[70] S. Feiner, B. MacIntyre, T. Hollerer, and A. Webster. A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment. In *IEEE International Symposium on Wearable Computers*, pages 74–81, October 1997.

[71] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[72] L. Florack, B. M. ter Haar Romeny, M. A. Viergever, and J. J. Koenderink. The Gaussian scale-space paradigm and the multiscale local jet. *International Journal of Computer Vision*, 18(1):61–75, April 1996.

[73] P. Fockler, T. Zeidler, B. Brombach, E. Bruns, and O. Bimber. PhoneGuide: museum guidance supported by on-device object recognition on mobile phones.

In *International Conference on Mobile and Ubiquitous Multimedia*, pages 3–10, December 2005.

[74] J. M. Frahm and M. Pollefeys. RANSAC for quasi-degenerate data (QDEGSAC). In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 453–460, June 2006.

[75] G. Francini, S. Lepsoy, and M. Balestri. *Telecom Italia Buildings Dataset*, November 2011. http://tinyurl.com/kaxucpb.

[76] G. Francini, S. Lepsoy, and M. Balestri. Selection of local features for visual search. *Signal Processing: Image Communications*, 28(4):311–322, April 2013.

[77] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions Mathematic Software*, 3(3):209–226, September 1977.

[78] A. Gelbukh, S. Han, and G. Sidorov. Compression of boolean inverted files by document reordering. In *IEEE Conference on Natural Language Processing and Knowledge Engineering*, pages 244–249, October 2003.

[79] B. Girod, V. Chandrasekhar, D. Chen, N.-M. Cheung, R. Grzeszczuk, Y. A. Reznik, G. Takacs, S. Tsai, and R. Vedantham. Mobile visual search. *IEEE Signal Processing*, 28(4):61–76, July 2011.

[80] B. Girod, V. Chandrasekhar, R. Grzeszczuk, and Y.A. Reznik. Mobile visual search: architectures, technologies, and the emerging MPEG standard. *IEEE MultiMedia*, 18(3):86 –94, March 2011.

[81] Google. Google Glass. http://www.google.com/glass.

[82] Google. Google Goggles, 2013. http://www.google.com/mobile/goggles.

[83] K. Grauman and T. Darrell. Efficient image matching with distributions of local invariant features. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 627–634, June 2005.

[84] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *IEEE International Conference on Computer Vision*, pages 1458–1465, October 2005.

[85] K. Grauman and T. Darrell. Approximate correspondences in high dimensions. In *Advances in Neural Information Processing Systems*, pages 505–512, December 2006.

[86] K. Grauman and T. Darrell. Pyramid match hashing: Sub-linear time indexing over partial correspondences. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

[87] R. M. Gray, K. L. Oehler, K. O. Perlmutter, and R. A. Ohlsen. Combining tree-structured vector quantization with classification and regression trees. In *IEEE Asilomar Conference on Signals, Systems and Computers*, pages 1494–1498, November 1993.

[88] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *ACM SIGMOD International Conference on Management of Data*, pages 47–57, 1984.

[89] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, pages 23.1–23.6, September 1988.

[90] T. Hassner, V. Mayzels, and L. Zelnik-Manor. On SIFTs and their scales. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1522–1528, June 2012.

[91] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Science, New York, NY, USA, 2009.

[92] M. Heikkila, M. Pietikainen, and C. Schmid. Description of interest regions with local binary patterns. *Pattern Recognition*, 42(3):425–436, March 2009.

[93] J. Heinly, E. Dunn, and J.-M. Frahm. Comparative evaluation of binary features. In *European Conference on Computer Vision*, pages 759–773, October 2012.

[94] A. Henrysson, M. Billinghurst, and M. Ollila. Face to face collaborative AR on mobile phones. In *IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 80–89, October 2005.

[95] S. Hussain and B. Triggs. Visual recognition using local quantized patterns. In *European Conference on Computer Vision*, pages 716–729, October 2012.

[96] Quest Visual Inc. Word Lens, 2013. http://questvisual.com/us/.

[97] M. Jain, H. Jegou, and P. Gros. Asymmetric Hamming embedding: taking the best of our bits for large scale image search. In *ACM International Conference on Multimedia*, pages 1441–1444, November 2011.

[98] M. Jamieson, S. Dickinson, S. Stevenson, and S. Wachsmuth. Using language to drive the perceptual grouping of local image features. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2102–2109, June 2006.

[99] H. Jegou and O. Chum. Negative evidences and co-occurrences in image retrieval: the benefit of PCA and whitening. In *European Conference on Computer Vision*, pages 774–787, October 2012.

[100] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *European Conference on Computer Vision*, pages 304–317, October 2008.

[101] H. Jegou, M. Douze, and C. Schmid. *INRIA Holidays Dataset*, June 2008. http://tinyurl.com/kkgmloh.

[102] H. Jegou, M. Douze, and C. Schmid. On the burstiness of visual elements. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1169–1176, June 2009.

[103] H. Jegou, M. Douze, and C. Schmid. Packing bag-of-features. In *IEEE International Conference on Computer Vision*, pages 2357–2364, September 2009.

[104] H. Jegou, M. Douze, and C. Schmid. Improving bag-of-features for large scale image search. *International Journal of Computer Vision*, 87(3):316–336, February 2010.

[105] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, January 2011.

[106] H. Jegou, M. Douze, C. Schmid, and P. Perez. Aggregating local descriptors into a compact image representation. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2010.

[107] H. Jegou, H. Harzallah, and C. Schmid. A contextual dissimilarity measure for accurate and efficient image search. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

[108] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid. Aggregating local image descriptors into compact codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1704–1716, September 2012.

[109] H. Jegou, C. Schmid, H. Harzallah, and J. Verbeek. Accurate image search using the contextual dissimilarity measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):2–11, January 2010.

[110] C. Jing, G. Junwei, and W. Yongtian. Mobile augmented reality system for personal museum tour guide applications. In *IET International Communication Conference on Wireless Mobile and Computing*, pages 262–265, November 2011.

[111] T. Kadir, A. Zisserman, and M. Brady. An affine invariant salient region detector. In *European Conference on Computer Vision*, pages 228–241, May 2004.

[112] L. Kantorovich. On the translocation of masses. *Management Science*, 5(1):1–4, October 1958.

[113] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *ACM SIGMOD International Conference on Management of Data*, pages 369–380, May 1997.

[114] N. Kingsbury. Rotation-invariant local feature matching with complex wavelets. In *European Conference on Signal Processing*, pages 4–8, September 2006.

[115] Kooaba. Kooaba Augmented Reality Engine, 2010. http://blog.kooaba.com/2010/04/mobile-augmented-reality-the-next-level-of-sophistication.

[116] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. Lopez, and J. V. B. Soares. Leafsnap: A computer vision system for automatic plant species identification. In *European Conference on Computer Vision*, pages 502–516, October 2012.

[117] D. Kurz and S. Ben Himane. Inertial sensor-aligned visual feature descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 161–166, June 2011.

[118] A. B. L. Larsen, S. Darkner, A. L. Dahl, and K. S. Pedersen. Jet-based local image descriptors. In *European Conference on Computer Vision*, pages 638–650, October 2012.

[119] S. Leutenegger, M. Chli, and R. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *IEEE International Conference on Computer Vision*, pages 2548–2555, November 2011.

[120] F.-F. Li and P. Perona. A Bayesian hierarchical model for learning natural scene categories. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 524–531, June 2005.

[121] J. Li. A source coding approach to classification by vector quantization and the principle of minimum description length. In *IEEE Data Compression Conference*, pages 382–391, March 2002.

[122] J. Li, N. Chaddha, and R. M. Gray. Multiresolution tree structured vector quantization. In *IEEE Asilomar Conference on Signals, Systems and Computers*, pages 922–925, November 1996.

[123] J. Lin, L.-Y. Duan, T. Huang, and W. Gao. *Peking University Buildings Dataset*, November 2011. http://tinyurl.com/mgr8fum.

[124] J. Lin, L.-Y. Duan, T. Huang, and W. Gao. Robust Fisher codes for large scale image retrieval. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1513–1517, May 2013.

[125] J. Lin, L.-Y. Duan, S. Yang, J. Chen, T. Huang, A. C. Kot, and W. Gao. Compact Descriptors for Visual Search: Performance improvements of the Scalable Compressed Fisher Vector. In *ISO/IEC JTCI/SC29/WG11 MPEG2013/M28061*, January 2013.

[126] Z. Lin and J. Brandt. A local bag-of-features model for large-scale object retrieval. In *European Conference on Computer Vision*, pages 294–308, September 2010.

[127] T. Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):77–116, February 1998.

[128] H. Ling and S. Soatto. Proximity distribution kernels for geometric context in category recognition. In *IEEE International Conference on Computer Vision*, pages 1–8, October 2007.

[129] H. Liu and S. Yan. Common visual pattern discovery via spatially coherent correspondences. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1609–1616, June 2010.

[130] X. Liu, Y. Lou, A. W. Yu, and B. Lang. Search by mobile image based on visual and spatial consistency. In *IEEE International Conference on Multimedia and Expo*, pages 1–6, July 2011.

[131] S. P. Lloyd. Least squares quantization in PCM. Technical report, July 1957.

[132] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[133] D. G. Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision*, pages 1150–1157, September 1999.

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

[135] M. Makar. *Interframe Compression of Visual Feature Descriptors for Mobile Augmented Reality*. PhD thesis, Stanford University, November 2013.

[136] M. Makar, S. Tsai, V. Chandrasekhar, D. Chen, and B. Girod. Interframe coding of canonical patches for mobile augmented reality. In *IEEE International Symposium on Multimedia*, pages 50–57, December 2012.

[137] M. Makar, S. Tsai, V. Chandrasekhar, D. Chen, and B. Girod. Interframe coding of canonical patches for low bit-rate mobile augmented reality. *International Journal of Semantic Computing*, 7(1):5–24, March 2013.

[138] S. Mann. Wearable computing: A first step toward personal imaging. *IEEE Computer*, 30(2):25–32, 1997.

[139] C. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[140] M. Marszaek and C. Schmid. Spatial weighting for bag-of-features. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2118–2125, June 2006.

[141] J. Matas, O. Chum, U. Martin, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *British Machine Vision Conference*, pages 36.1–36.10, September 2002.

[142] Meta. Meta Metapro. https://www.spaceglasses.com.

[143] K. Mikolajczyk and C. Schmid. An affine invariant interest point detector. In *European Conference on Computer Vision*, pages 128–142, May 2002.

[144] K. Mikolajczyk and C. Schmid. Scale and affine invariant interest point detectors. *International Journal of Computer Vision*, 60(1):63–86, October 2004.

[145] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, October 2005.

[146] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool. A comparison of affine region detectors. *International Journal of Computer Vision*, 65(1-2):43–72, November 2005.

[147] A. Mikulik, M. Perdoch, O. Chum, and J. Matas. Learning a fine vocabulary. In *European Conference on Computer Vision*, pages 1–14, September 2010.

[148] T. Miyashita, P. Meier, T. Tachikawa, S. Orlic, T. Eble, V. Scholz, A. Gapel, O. Gerl, S. Arnaudov, and S. Lieberknecht. An augmented reality museum guide. In *IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 103–106, September 2008.

[149] A. Moffat and V. N. Anh. Binary codes for non-uniform sources. In *IEEE Data Compression Conference*, pages 133–142, March 2005.

[150] A. Moffat and V. N. Anh. Binary codes for locally homogeneous sequences. *Information Processing Letters*, 99(5):175–180, September 2006.

[151] A. Moffat and L. Stuiver. Exploiting clustering in inverted file compression. In *IEEE Data Compression Conference*, pages 82–91, March 1996.

[152] A. Moffat and L. Stuiver. Binary interpolative coding for effective index compression. *Information Retrieval*, 3(1):25–47, July 2000.

[153] A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, October 1996.

[154] H. Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. In *Technical Report CMU-RI-TR-80-03, Robotics Institute, Carnegie Mellon University*, number CMU-RI-TR-80-03. September 1980.

[155] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application*, pages 331–340, February 2009.

[156] K. Ni, H. Jin, and F. Dellaert. GroupSAC: Efficient consensus in the presence of grouping. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2193–2200, June 2009.

[157] D. Nister. Preemptive RANSAC for live structure and motion estimation. In *IEEE International Conference on Computer Vision*, pages 199–206, October 2003.

[158] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2161–2168, June 2006.

[159] D. Nister and H. Stewenius. Linear time maximally stable extremal regions. In *European Conference on Computer Vision*, pages 183–196, October 2008.

[160] Nokia. Nokia Point-and-Find, 2009. http://europe.nokia.com/services-and-apps/nokia-point-and-find.

[161] E. Nowak, F. Jurie, and B. Triggs. Sampling strategies for bag-of-features image classification. In *European Conference on Computer Vision*, pages 490–503, May 2006.

[162] T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987, July 2002.

[163] I. Olonetsky and S. Avidan. TreeCANN: k-d tree coherence approximate nearest neighbor algorithm. In *European Conference on Computer Vision*, pages 602–615, October 2012.

[164] K. M. Ozonat. Image classification using tree-structured discriminant vector quantization. In *IEEE Asilomar Conference on Signals, Systems and Computers*, pages 1610–1614, November 2003.

[165] K. M. Ozonat and R. M. Gray. Image classification using adaptive-boosting and tree-structured discriminant vector quantization. In *IEEE Data Compression Conference*, page 556, March 2004.

[166] B. Parhizkar, Z.M. Gebril, W.K. Obeidy, M.N.A. Ngan, S.A. Chowdhury, and A.H. Lashkari. Android mobile augmented reality application based on different learning theories for primary school children. In *International Conference on Multimedia Computing and Systems*, pages 404–408, May 2012.

[167] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

[168] F. Perronnin, Y. Liu, J. Sanchez, and H. Poirier. Large-scale image retrieval with compressed Fisher vectors. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3384–3391, June 2010.

[169] F. Perronnin, J. Sanchez, and T. Mensink. Improving the Fisher kernel for large-scale image classification. In *European Conference on Computer Vision*, pages 143–156, September 2010.

[170] M. Persin. Document filtering for fast ranking. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 339–348, July 1994.

[171] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

[172] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008.

[173] J. Philbin, J. Sivic, and A. Zisserman. Geometric latent dirichlet allocation on a matching graph for large-scale image datasets. *International Journal of Computer Vision*, 95(2):138–153, November 2011.

[174] J. Philbin and A. Zisserman. *Oxford Buildings Dataset*, June 2007. http://tinyurl.com/2afuglg.

[175] T. Quack, V. Ferrari, and L. V. Gool. Video mining with frequent itemset configurations. In *International Conference on Image and Video Retrieval*, pages 360–369, July 2006.

[176] Y. Reznik, G. Cordara, and M. Bober. Evaluation framework for Compact Descriptors for Visual Search. In *ISO/IEC JTCI/SC29/WG11 N12202*, July 2011.

[177] E. A. Riskin and R. M. Gray. A greedy tree growing algorithm for the design of variable rate vector quantizers. *Signal Processing, IEEE Transactions on*, 39(11):2500–2507, November 1991.

[178] J. T. Robinson. The K-D-B-tree: A search structure for large multidimensional dynamic indexes. In *ACM SIGMOD International Conference on Management of Data*, pages 10–18, April 1981.

[179] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, pages 430–443, May 2006.

[180] E. Rosten, R. Porter, and T. Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):105–119, January 2010.

[181] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *IEEE International Conference on Computer Vision*, pages 2564–2571, November 2011.

[182] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. In P. Willett, editor, *Document Retrieval Systems*, pages 132–142. 1972.

[183] A. Said. Comparative analysis of arithmetic coding computational complexity. In *HP Labs Technical Report*, 2004.

[184] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[185] J. Sanchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1665–1672, June 2011.

[186] T. Sattler, B. Leibe, and L. Kobbelt. SCRAMSAC: Improving RANSAC's efficiency with a spatial consistency filter. In *IEEE International Conference on Computer Vision*, pages 2090–2097, September 2009.

[187] G. Schindler, M. Brown, and R. Szeliski. City-scale location recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7, June 2007.

[188] F. Scholer, H. E. Williams, J. Yiannis, and J. Zobel. Compression of inverted indexes for fast query evaluation. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 222–229, August 2002.

[189] C. Seifert, L. Paletta, A. Jeitler, E. Hodl, and J.-P. Andreu. Visual object detection for mobile road sign inventory. In *International Conference on Human Computer Interaction with Mobile Devices and Services*, pages 491–495, September 2004.

[190] H. Shao, T. Svoboda, and L. V. Gool. *ZuBuD: Zurich Buildings Dataset*, April 2003. http://tinyurl.com/6ctwwl6.

[191] J. Shi and C. Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, June 1994.

[192] W. Y. Shieh, T. F. Chen, J. J. Shann, and C. P. Chung. Inverted file compression through document identifier reassignment. *Information Processing and Management*, 39(1):117–131, January 2003.

[193] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *IEEE International Conference on Computer Vision*, pages 1470–1477, October 2003.

[194] J. Sivic and A. Zisserman. Video data mining using configurations of viewpoint invariant regions. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 488–495, June 2004.

[195] A. Smailagic and D.P. Siewiorek. The CMU mobile computers: A new generation of computer systems. In *IEEE Computer Society International Conference*, pages 467–473, February 1994.

[196] S. M. Smith and J. M. Brady. SUSAN: A new approach to low level image processing. *International Journal of Computer Vision*, 23(1):45–78, May 1997.

[197] R. F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6(1-6):579–589, 1991.

[198] H. Stewenius and D. Nister. *University of Kentucky Benchmark Dataset*, September 2006. http://tinyurl.com/ddoht2.

[199] I. Sutherland. The ultimate display. In *Proceedings of International Federation of Information Processing*, volume 2, pages 506–508. 1965.

[200] I. Sutherland. A head-mounted three dimensional display. In *Joint Computer Conference*, pages 757–764, December 1968.

[201] G. Takacs. *Unified Tracking and Recognition with Rotation-Invariant Fast features*. PhD thesis, Stanford University, February 2012.

[202] G. Takacs, V. Chandrasekhar, N. Gelfand, Y. Xiong, W.-C. Chen, T. Bismpigiannis, R. Grzeszczuk, K. Pulli, and B. Girod. Outdoors augmented reality on mobile phone using loxel-based visual feature organization. In *ACM Multimedia Information Retrieval*, pages 427–434, October 2008.

[203] G. Takacs, V. Chandrasekhar, S. Tsai, D. Chen, R. Grzeszczuk, and B. Girod. Unified real-time tracking and recognition with rotation-invariant fast features. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 934–941, June 2010.

[204] G. Takacs, V. Chandrasekhar, S. Tsai, D. Chen, R. Grzeszczuk, and B. Girod. Rotation-invariant fast features for large-scale recognition and real-time tracking. *Signal Processing: Image Communication*, 28(4):334–344, April 2013.

[205] E. Tola, V. Lepetit, and P. Fua. A fast local descriptor for dense matching. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008.

[206] P. H. S. Torr and A. Zisserman. MLESAC: a new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, April 2000.

[207] S. Tsai, D. Chen, G. Takacs, V. Chandrasekhar, J. P. Singh, and B. Girod. Location coding for mobile image retrieval. In *International Conference on Mobile Multimedia Communications*, pages 1–7, September 2009.

[208] S. Tsai, D. Chen, G. Takacs, V. Chandrasekhar, R. Vedantham, R. Grzeszczuk, and B. Girod. Fast geometric re-ranking for image-based retrieval. In *IEEE International Conference on Image Processing*, pages 1029–1032, September 2010.

[209] S. Tsai, H. Chen, D. Chen, and B. Girod. WORD-HOGS: Word histogram of oriented gradients for mobile visual search. In *IEEE International Conference on Image Processing*, September 2014. Submitted.

[210] T. Tuytelaars. Dense interest points. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2281–2288, June 2010.

[211] T. Tuytelaars and C. Schmid. Vector quantizing feature space with a regular lattice. In *IEEE International Conference on Computer Vision*, pages 1–8, October 2007.

[212] T. Tuytelaars and L. V. Gool. Matching widely separated views based on affine invariant regions. *International Journal of Computer Vision*, 59(1):61–85, August 2004.

[213] F. von Hundelshausen. D-Nets: Beyond patch-based image descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2941–2948, June 2012.

[214] X. Wang, M. Yang, T. Cour, S. Zhu, K. Yu, and T. X. Han. Contextual weighting for vocabulary tree based image retrieval. In *IEEE International Conference on Computer Vision*, pages 209–216, November 2011.

[215] C. Wengert, M. Douze, and H. Jegou. Bag-of-colors for improved image search. In *ACM International Conference on Multimedia*, pages 1437–1440, November 2011.

[216] S. Winder, G. Hua, and M. Brown. Picking the best DAISY. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 178–185, June 2009.

[217] Z. Wu, Q. Ke, M. Isard, and J. Sun. Bundling features for large scale partial-duplicate web image search. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 25–32, June 2009.

[218] Z. Wu, Q. Ke, J. Sun, and H.-Y. Shum. A multi-sample, multi-tree approach to bag-of-words image representation for image retrieval. In *IEEE International Conference on Computer Vision*, pages 1992–1999, September 2009.

[219] S. Yan, X. Xu, D. Xu, S. Lin, and X. Li. Beyond spatial pyramids: A new feature extraction framework with dense spatial sampling for image classification. In *European Conference on Computer Vision*, pages 473–487, October 2012.

[220] T. Yeh, J. J. Lee, and T. Darrell. Adaptive vocabulary forests for dynamic indexing and category learning. In *IEEE International Conference on Computer Vision*, pages 1–8, October 2007.

[221] J. Yuan, Y. Wu, and M. Yang. Discovery of collocation patterns: from visual words to visual phrases. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

[222] J. Zhang, X. Long, and T. Suel. Performance of compressed inverted list caching in search engines. In *International Conference on World Wide Web*, pages 387–396, April 2008.

[223] S. Zhang, Q. Huang, G. Hua, S. Jiang, W. Gao, and Q. Tian. Building contextual visual vocabulary for large-scale image applications. In *ACM International Conference on Multimedia*, pages 501–510, October 2010.

[224] X. Zhang, Z. Li, L. Zhang, W.-Y. Ma, and H.-Y. Shum. Efficient indexing for large scale visual search. In *IEEE International Conference on Computer Vision*, pages 1103–1110, September 2009.

[225] Y. Zhang, Z. Jia, and T. Chen. Image retrieval with geometry-preserving visual phrases. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 809–816, June 2011.

[226] L. Zheng, S. Wang, Z. Liu, and Q. Tian. Lp-norm IDF for large scale image search. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1626–1633, June 2013.

[227] W. Zhou, H. Li, Y. Lu, and Q. Tian. Large scale image search with geometric coding. In *ACM International Conference on Multimedia*, pages 1349–1352, November 2011.

[228] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2):1–56, July 2006.

[229] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23(4):453–490, December 1998.