Efficient Retrieval of Web Content with Distinctive Visual Vocabularies

Kastrinakis Dimitrios (335)
1/19/2012

This thesis is submitted in partial fulfillment of the requirements for the Degree of Master of Science in Information Systems at the Aristotle University of Thessaloniki.

Supervisor: Athena Vakali, Associate Professor
## Contents

Table of Figures ................................................................. 4  
Abstract ............................................................................. 7  
Περιλήψη ............................................................................. 8  
Acknowledgements ................................................................ 9  
1 Introduction ..................................................................... 10  
   1.1 Image Retrieval ......................................................... 10  
   1.2 Indexing an Image Collection .................................... 10  
   1.3 Main Contributions .................................................. 11  
2 Related Work .................................................................. 13  
   2.1 Object Retrieval of Video Frames ............................... 13  
   2.2 Quantization of the Features ..................................... 13  
      2.2.1 Hierarchical K-means .......................................... 13  
      2.2.2 Approximate K-means with k-d trees .................... 14  
   2.3 Randomized k-d trees Vs Hierarchical K-means tree .... 15  
   2.4 Locality Sensitive Hashing ......................................... 16  
   2.5 Soft-Assignment ...................................................... 16  
   2.6 Query Expansion ..................................................... 17  
   2.7 Binary Codes .......................................................... 18  
   2.8 Product Quantization ............................................... 18  
   2.9 Efficient Similarity Search with Graphs .................... 21  
3 Building a More Distinctive Visual Vocabulary .................. 23  
   3.1 The Feature Set ....................................................... 23  
      3.1.1 Overview of SIFT Keypoints and Descriptors ....... 23  
      3.2 Quantization of the Feature Set ............................... 25  
      3.2.1 Clustering Large Volumes of Data ....................... 25  
      3.2.2 Avoiding a Poor Clustering with K-means++ .......... 26  
      3.2.3 Lowering the Time Required for Quantization ....... 28  
   3.3 Indexing the Collection with Composite Visual Words .... 28  
      3.3.1 The Bag-of-visual-words Model ......................... 28  
      3.3.2 Composite Visual Words .................................... 30  
      3.3.3 Distance Thresholds .......................................... 32  
      3.3.4 Indexing with Apache Solr ................................. 33  
   3.4 Query Processing ..................................................... 34
Table of Figures

Figure 1.1: Main steps followed in order to index a collection of images for future retrieval 11
Figure 2.1: building the vocabulary tree with a k-means process recursively applied to each group of descriptor vectors (image appeared in [39]) .................................................................................................................. 14
Figure 2.2: k-d tree partitions of a uniform set of data points using the median (image appeared in http://www.vfeat.org/overview/kdtree.html) ........................................................................................................... 15
Figure 2.3 with soft-assignment, a descriptor can be assigned to a combination of visual words (image appeared in [45]) ................................................................................................................... 17
Figure 2.4 Symmetric (SDC) and Asymmetric (ADC) distance computation between two vectors (image appeared in [27]) ................................................................................................................ 19
Figure 2.5: insertion and search procedures in the inverted file (image appeared in [27]) ... 20
Figure 2.6: distinguishing between landmarks and events with a hybrid image similarity graph (image appeared in [42]). ........................................................................................................... 21
Figure 2.7: similarity graph generated from the top-1000 search results of "Mona-Lisa". The largest two images are the most important, as they contain the highest rank (image appeared in [29]). .......................................................................................................... 22
Figure 3.1 a SIFT keypoint is characterized by its location, scale and orientation......... 23
Figure 3.2: for each of the 16 4x4 windows, 16 gradient orientations are put into an 8-bin histogram (image appeared in http://www.vlfeat.org/overview/kdtree.html)................................................................. 24
Figure 3.3: scheme for storing the feature sets of a collection of images.......................... 24
Figure 3.4: a streaming divide and conquer version of k-means ....................................... 26
Figure 3.5: a suboptimal k-means clustering ........................................................................ 27
Figure 3.6: k-means clustering can be arbitrarily suboptimal ............................................. 27
Figure 3.7: In the bag-of-visual-words model, for each feature of an image, the label of closest cluster center is kept, forming a document of visual words which will be indexed as in text retrieval................................................................................................. 29
Figure 3.8: an inverted index.............................................................................................. 30
Figure 3.9: composing visual words based on the labels of the nearest cluster centers...... 30
Figure 3.10: a Voronoi diagram in 2D space........................................................................ 31
Figure 3.11: a document retrieved from Solr server as a result of a query ....................... 34
Figure 3.12: the main window of the user interface............................................................. 35
Figure 3.13: the configuration window of the user interface ............................................. 36
Figure 4.1: images from the Oxford buildings dataset, each depicting a different landmark 38
Figure 4.2: mAP results for increasing initial vocabulary size ........................................... 39
Figure 4.3: top-5 results of the top-3 queries for k=3 and |V|=200 (the first row depicts the query images) ................................................................................................................................. 41
Figure 4.4: results of the bottom-3 queries for k=3 and |V|=200 (the first row depicts query images) ................................................................................................................................. 42
Figure 4.5: histograms of AP score frequencies .................................................................... 43
Figure 4.6: index size for increasing initial vocabulary |V| ..................................................... 44
Figure 4.7: indexing time for increasing initial vocabulary |V| ............................................... 45
Figure 4.8: average time required to process each of the 55 queries ............................... 46
Figure 4.9: images from the Paris buildings dataset, each depicting a different landmark... 47
Figure 4.10: mAP results for increasing vocabulary size ................................................... 48
Figure 4.11: top-5 results of the top-3 queries for k=3 and |V|=200 (the first row depicts the query images) ................................................................. 49
Figure 4.12: results of the bottom-3 queries for k=3 and |V|=200 (the first row depicts query images) ........................................................................ 50
Figure 4.13: histograms of AP score frequencies ................................................................. 51
Figure 4.14: index size for increasing initial vocabulary |V| ........................................... 52
Figure 4.15: indexing time for increasing initial vocabulary |V| ........................................... 52
Figure 4.16: average time required to process each of the 55 queries .............................. 53

Table 1: number of images and SIFT vectors extracted from the Oxford Buildings dataset .. 39
Table 2: comparison between our method and exact k-means, hierarchical (HKM) and approximate (AKM) k-means, as measured in [44] .............................................. 40
Table 3: top and bottom 3 queries for k=3 and |V|=200 ............................................... 41
Table 4 ........................................................................................................... 44
Table 5: number of images and SIFT vectors extracted from the Paris Buildings dataset .... 47
Table 6: top and bottom 3 queries for k=3 and |V|=200 ............................................... 48
Table 7 ........................................................................................................... 52

Code 1: saving and loading the extracted feature set...................................................... 25
Code 2: our implementation of the streaming divide and conquer for k-means .......... 26
Code 3: Algorithm that forms composite visual words ............................................... 33
Abstract
Image retrieval is a constantly evolving field of research, proposing new solutions and improving existing ones as users have access to massive volumes of both data and metadata on the Web. Technologies for the Web 2.0 allow users to create and publish their own information and share it with online friends, instead of just consuming information, which had been true for the previous decade. For example, Flickr hosts billions of user uploaded pictures, available for searching, browsing, tagging and sharing; indexing such a massive collection of images is usually an arduous and non-trivial task.

Building an index for future retrieval based on the bag of words model consists of three major tasks: (a) local feature extraction from a collection of images and storage to the file system or a database. Each image is provided as input to an appropriate algorithm which extracts a set of descriptors. A local descriptor is a multidimensional vector which describes in a distinctive manner the point of interest (keypoint) it belongs to. The keypoint is a small area of the image which surrounds an edge or a corner. In the case of global features, a single descriptor is extracted from each image and it may describe color, texture or shape. (b) Quantization of the feature space to produce visual words, thus building the visual vocabulary. Often, this can be achieved by applying a clustering method to the feature set such as k-means, which is an expensive process, since the feature vectors can amount to millions. (c) Indexing each image in the collection as a document of visual words, reducing image retrieval to a text retrieval problem. For each feature vector in an image, the nearest visual word is kept, resulting in a bag of visual words for each image. However, (c) is not always adopted; there are other approaches that rely on a histogram representation of images and use the Euclidean distance between vectors of images.

In this thesis, we propose a novel method that significantly decreases the time required for the quantization process. Instead of generating hundreds of thousands of visual words during the quantization step, only a few visual words suffice: multiple visual words can be assigned to a feature vector in ascending order of distance, as long as they pass an appropriately selected distance threshold. In this way, a server can host even larger collections and respond to queries in less time, while preserving an adequate retrieval quality. We evaluate our system using reference collections of social content by calculating the mean average precision of the proposed method and comparing it to the baseline. Moreover, ranked results for specific queries are discussed and the time consumed for indexing and querying is measured.
Περίληψη
Η ανάκτηση εικόνων αποτελεί ένα διαρκώς αναπτυσσόμενο πεδίο έρευνας, προτείνοντας νέες λύσεις και βελτιώνοντας τις ήδη υπάρχουσες καθώς οι χρήστες αποκτούν πρόσβαση σε μεγάλο όγκο δεδομένων και μετα-δεδομένων στον Παγκόσμιο Ιστό. Τεχνολογίες για το Web 2.0 επιτρέπουν τη δημιουργία, δημοσίευση και διαμοιρασμό πληροφορίας μεταξύ συνδεδεμένων φίλων, αντί για την παθητική κατανάλωση πληροφορίας που ισχύει μέχρι τις αρχές της προηγούμενης δεκαετίας. Για παράδειγμα, το Flickr φιλοξενεί δισεκατομμύρια φωτογραφίες χρηστών, διαθέσιμες για αναζήτηση, επισκόπηση, προσάρτηση μετα-δεδομένων (tagging) και διαμοιρασμό. Ο ευρετηριασμός μιας συλλογής εικόνων τέτοιου μεγέθους είναι συνήθως δύσκολο και μη τετριμμένο εγχείρημα.

Η κατασκευή ενός ευρετηρίου προς μελλοντική ανάκτηση βάσει του μοντέλου bag-of-visual-words υλοποιείται σε τρία βασικά βήματα: (α) Την εξαγωγή τοπικών χαρακτηριστικών (local features) από μια συλλογή εικόνων και αποθήκευσή τους στο σύστημα αρχείων ή σε βάση δεδομένων. Κάθε εικόνα δίδεται ως είσοδος σε έναν κατάλληλο αλγόριθμο και εξάγεται ένα σύνολο από περιγραφές (descriptors). Ένας τοπικός περιγραφέας είναι ένα πολυδιάστατο διάνυσμα που περιγράφει με χαρακτηριστικό τρόπο το σημείο ενδιαφέροντος όπου ανήκει. Ένα σημείο ενδιαφέροντος είναι μια μικρή περιοχή που περικλείει κάποια άκρη ή γωνία στην εικόνα. (β) Την κβάντιση (quantization) του χώρου των χαρακτηριστικών για την παραγωγή «οπτικών λέξεων» (visual words) και τελικά την κατασκευή του «οπτικού λεξιλογίου» (visual vocabulary). Συχνά, αυτό επιτυγχάνεται με την εφαρμογή μεθόδους ομαδοποίησης στο σύνολο των δεδομένων των χαρακτηριστικών, όπως ο k-means, η οποία είναι μια ακριβή διαδικασία, καθώς τα διανύσματα χαρακτηριστικών μπορεί εύκολα να ανέλθουν σε εκατομμύρια. (γ) Την ευρετηρίαση (indexing) κάθε εικόνας στη συλλογή ως ένα έγγραφο οπτικών λέξεων, ανάγοντας το εν λόγω πρόβλημα σε ανάκτηση κειμένου. Για κάθε χαρακτηριστικό στην εικόνα, επιλέγεται η εγγύτερη οπτική λέξη, εν τέλει διαμορφώνοντας ένα σάκο από οπτικές λέξεις για κάθε εικόνα. Βεβαίως, το (γ) δεν υιοθετείται πάντοτε: υπάρχουν άλλες προσεγγίσεις που βασίζονται σε παραγόμενα ιστογράμματα για κάθε εικόνα, χρησιμοποιώντας την Ευκλείδεια απόσταση στην ανάκτηση.
Acknowledgements

I would like to thank my supervisor, Associate Professor Athena Vakali, for her support and confidence in me.

Of course, I want to thank the other two members of the examination committee, Associate Professor Eleftherios Angelis and Lecturer Grigorios Tsoumakas for the evaluation of this work.

I would also like to express my sincere gratitude to Dr. Symeon Papadopoulos for our excellent cooperation, whose vital counsel and helpful guidance significantly contributed to the completion of this thesis.

Finally, it would be ungrateful not to express my deepest appreciation and most heartfelt thanks to my family for their endless support and understanding, as well as their unfathomable patience.
1 Introduction

1.1 Image Retrieval
Image Retrieval can be divided into two main categories: image retrieval using annotations and content based image retrieval. In the case of annotations, a search is performed based on associated metadata such as tags, description, etc. For example, Flickr is a popular host of user photographs and videos, allowing the user to provide information such as title, location, description, etc. about the web content they share. However, a search engine of this type would require extra space for storing metadata and it would introduce a variety of complications: an annotation is valid only for one language, and subjectivity of human perception hides some aspects of the image from the user. Also, it requires too much work from the end user; tagging and describing a whole album of photographs before uploading is not a negligible task.

On the other hand, content-based image retrieval (CBIR) is the application of computer vision techniques to the image retrieval problem, tackling the problems of ambiguity and subjectivity from which annotations suffer. CBIR retrieves images from a digital database based on similarities in their contents to a user supplied image query. An Image Retrieval system of this type allows browsing, searching, ranking and retrieving images from a large database of digital images, either ignoring annotations or taking them into account. Object Retrieval refers to the field of research where, given a particular object in an image, relevant images that depict the specified object are retrieved, ranked and presented to the user. In practice, there is no perfect binary function that determines relevance in images; even human judges may disagree on the relevance of images due to subjectivity or technical factors regarding the image quality such as occlusion, distortion, different lighting, etc. Furthermore, the semantic gap between low-level features and high-level semantic concepts is a fundamental subject in image retrieval and strenuous efforts have been made towards bridging it [61, 56].

However, recent work in image retrieval shows a serious effort in reducing this problem to the context of text retrieval using the analogy of “visual words”, indexing each image in the collection as a document or a bag of visual words. The bag of words model is a simplifying assumption where text is represented as an unordered collection of words, disregarding grammar and word order. In this way, retrieval from an image collection can take place with reuse of known technology and successful methods applied to classic text retrieval.

1.2 Indexing an Image Collection
Indexing a collection of images consists of three main steps, as illustrated in Figure 1.1: at first, a feature set is extracted from the image collection. An image is scanned for points of interest using popular methods such as SIFT or SURF [33, 18]. Each point of interest, also known as keypoint, is an image region characterized by a location, scale, orientation and a multidimensional vector. This descriptor vector is the fingerprint of the keypoint so that similar vectors belong to similar keypoints. Similarity and distance between vectors can be computed with various measures like cosine similarity or the Euclidean distance. Substantial engineering effort has been devoted to the study of feature detection and description; in
this work, we extracted 128-D SIFT descriptors, which provide robust image matching with scale, rotation and illumination invariance.

Secondly, a quantization process follows. This can be achieved by clustering the feature set in order to produce the visual vocabulary, where each cluster center found represents a visual word. In this way, identical keypoints, or keypoints that share similar visual characteristics can be assigned to the same visual word, resembling a word in the textual context, which is used to describe an object, concept or idea. While simple k-means is probably the most common clustering algorithm, there are more sophisticated approaches [37, 21, 30, 50], such as Hierarchical k-means (HKM) [39], in which k-means is recursively applied to the clusters and creates more refined partitions of the data.

Last, images in the collection are indexed as documents/bags of visual words, building an inverted index on the basis of a classic TF-IDF [60] scheme. A visual word corresponds to the label of the cluster center found nearest to a feature vector. The arising issue is the need of a rich and distinctive vocabulary. To this end, an expensive clustering process must be completed at the second step. For example, k-means is linear to the number of vectors given as input, their dimensionality and the desired number of centers. Since a satisfactory vocabulary can contain up to a 1M words, it is imperative to substantially decrease the time consumed during quantization.

![Figure 1.1: Main steps followed in order to index a collection of images for future retrieval](image)

### 1.3 Main Contributions

It would be useful to define the nearest neighbor problem, on which our main contribution partly relies. Given a set of points $P = \{p_1, p_2, ..., p_n\}$ in vector space $X$, a positive integer $k$ and a query point $q \in X$, the nearest neighbor query $NN(q, k)$ selects the $k$ points lying nearest to $q$. Due to the high dimensionality of the feature space, approximate nearest neighbors search can be no more efficient than linear search, which is too costly, especially given the massive image datasets of created in the context of social media applications. This led to developing approximate nearest neighbor algorithms capable of searching orders of magnitude faster and providing near optimal retrieval [15, 32, 51, 38, 17]. We will focus on exact neighbors, but with an extremely smaller set of points. For instance, contrary to a vocabulary of 100K words, we propose a method that requires only a few hundred visual words, reducing the time required for quantization by three orders of magnitude: tests reveal that as few as 200 cluster centers can be utilized to build a retrieval system of adequate precision and recall. Moreover, we observe significant performance gains on building the index of the collection.
The main contribution of our work lies in the assignment of multiple visual words to each descriptor, instead of only one. The assigned visual words are the $k$ nearest centers to the descriptor that qualify our proposed distance threshold, which are then concatenated in ascending order of distance, forming a **composite visual word**, i.e. a word that consists of more than one visual words. In other words, a feature vector is described by a permutation of multiple visual words instead of only one, resulting in an enriched visual vocabulary. For the same initial number of visual words, our experiments show that the precision and recall of our approach is significantly better than the standard approach. For the same precision and recall, our approach consumes only a fraction of the standard quantization and indexing requirements.

Having built the visual vocabulary and the inverted index, the retrieval system accepts user provided images as queries. Then it extracts the feature set from the image query and determines the nearest visual words for each feature vector, formulating composite visual words as query terms. The query is processed as a document of visual words and ranked results are presented to the user, ordered by similarity.

To examine the performance of our retrieval system, we indexed images from known datasets (Oxford Buildings [1] and Paris Buildings [2]) often used for such purposes. Then we evaluated the retrieval accuracy by calculating the mean average precision (mAP) for a set of 55 queries with an associated ground truth.

Our work can be outlined as follows: in section 2 we will cover important contributions in the area of image retrieval and quantization of the feature space. Section 3 provides a thorough description of our approach and a detailed evaluation follows in section 4. Finally, in section 5 we summarize our findings and consider some future work as a further improvement for our proposed system.
2 Related Work

Most of the work in content based image/object retrieval is based on the bag-of-visual words approach. According to this approach, descriptor vectors are extracted from the collection of images, which are then quantized to produce the cluster centers of the feature space \([35, 39, 52]\). These centers, also known as visual words in the context of CBIR, form the visual vocabulary. Each image is presented as a bag of visual words, using a function that maps the high dimensional descriptors into the words of the vocabulary. Finally, indexing is performed on the collection, borrowing methodologies from the text retrieval literature \([60]\). This section overviews important work on the bag of visual words model and image retrieval in general.

2.1 Object Retrieval of Video Frames

One of the earliest attempts towards object retrieval systems based on text retrieval is described in \([52]\). Sivic and Zisserman introduce a bag-of-visual-words approach that searches for all the occurrences of a user indicated object in a video. The descriptor vectors, which are computed for each frame, are then quantized into visual words using k-means clustering. For each visual word there is an entry which stores all the occurrences of the same word in the video frames, thus building an inverted file. Each frame in the video is represented by a vector of visual words, employing a term frequency-inverse document frequency (TF-IDF) weighting on the components of this vector. A query vector is provided by the visual words contained in a user specified part of a frame, and frames are ranked according to the similarity of their vectors to the query vector.

Text retrieval systems often promote documents where the query keywords appear close together \([44, 52]\). Sivic and Zisserman implement an analogy for object retrieval, where matched regions in the retrieved frames of a video should have similar spatial arrangement to the regions of the query image. The idea is that neighboring matches in the query region lie in a surrounding area in the retrieved frame, which is implemented by defining an area of 15 nearest neighbors of each match. Each region which also matches within this area casts a vote for that frame, and the total votes determine the rank of the frame in the list of results.

2.2 Quantization of the Features

2.2.1 Hierarchical K-means

An approach inspired by the above work is proposed in \([39]\). In that paper, Nistér and Stewéníus introduce a recognition scheme in which local region descriptors are hierarchically quantized in a vocabulary tree. The hierarchically defined visual words, used for TF-IDF scoring, allow more efficient lookup of words, enabling the use of a larger vocabulary which improves the retrieval quality. In particular, this tree is built by hierarchical k-means clustering (HKM) using a set of descriptor vectors for the unsupervised creation of the tree.
Instead of $k$ defining the desired number of clusters, $k$ defines the branch factor, i.e. the number of children of each node. An initial k-means process clusters the training data to $k$ groups, where each group consists of the descriptor vectors closest to its center. As illustrated in Figure 2.1, this process is recursively applied to each group, forming a specified maximum number of levels $L$. This results in $k^L$ clusters, and each node in the tree represents a cluster center. A descriptor vector is propagated down the tree by comparing the vector to the cluster centers that reside at each level. Eventually, $kL$ comparisons are performed in order to reach a leaf node. The computational cost of increasing the size of the hierarchical vocabulary is logarithmic to the number of leaf nodes, and the memory usage is linear to the number of leaf nodes $k^L$.

### 2.2.2 Approximate K-means with k-d trees

Another clustering method often used for feature space quantization is the approximate k-means clustering (AKM) [44]. Typical k-means implementations fail to scale to large volumes of data, since the time complexity of each iteration is linear to the number of data, dimensionality of the data and the number of desired clusters. Instead of calculating the exact nearest neighbors between data points and cluster centers, an approximate nearest neighbor method can be applied to increase speed. In [44], a forest of eight randomized k-d trees is used [31], which is built over the cluster centers at the beginning of each iteration. In a typical k-d tree, nodes represent k-dimensional points and every non-leaf node splits space in two parts, using the dimension with the highest variance. The value to split on is the median or the mean of the dimension. Figure 2.2 shows the resulting partition of space using a k-d tree. In a randomized k-d tree, the splitting dimension is chosen at random among a set of the dimensions with the highest variance. The splitting value is also randomly chosen, but close to the median. A forest of randomized k-d trees prevents points lying close to partition boundaries from being assigned to an incorrect nearest neighbor. This is especially important in quantization of high dimensional features, such as 128-D SIFT descriptors.
A data point (in this case, a descriptor) is assigned to the approximately closest cluster center by descending to a leaf and storing the distances to the boundaries in a single priority queue \([17]\) for all trees. The most suitable branch is chosen iteratively and unexplored nodes are added to the queue. Once a fixed number of paths have been explored, the operation stops.

Approximate k-means with k-d trees reduces the complexity each iteration from \(O(ndc)\) to \(O(nd \log c)\), where \(n\) is the number of features, \(d\) their dimensionality and \(c\) the number of clusters. Philbin et al. showed that the percentage of points assigned to different cluster centers using approximate k-means differs from the exact version by less than 1% \([44]\). However, for high dimensions, the k-d tree is not more efficient than the brute force exhaustive distance calculation, which has \(O(nd)\) complexity \([57]\).

### 2.3 Randomized k-d trees Vs Hierarchical K-means tree

In \([38]\), Muja and Lowe compare two data structures for approximate nearest neighbors, the randomized k-d trees and the hierarchical k-means tree, after modifying the latter by adding a priority queue. After a traversal, the branches of each explored node are added to the queue, and the branch containing the closest center to the query point is extracted from the queue, initiating a new traversal from that branch. In each traversal, the unexplored branches along the path are added to the queue, and the search stops after a predetermined number of leaves (dataset points) have been examined. This number specifies a degree of approximation in the same way as for the randomized k-d trees.

In addition, they implemented an algorithm that determines the fastest approximate nearest neighbor algorithm for a specified dataset. The user provides the desired degree of precision, certain weights for the build time, search time and memory consumption and the most appropriate ANN algorithm is returned, along with its configuration. The algorithm configuration includes parameters such as branching factor and iterations in the case of the hierarchical k-means tree, and the number of trees in the case of the forest of randomized k-d trees. Muja and Lowe concluded that each method is appropriate depending on the
dataset and the user requirements; however, both randomized k-d trees and the hierarchical k-means tree outperform the ANN and Locality Sensitive Hashing (mentioned in 2.4) algorithms proposed in [17] and [15], respectively.

Muja and Lowe also discovered that the speedup over linear search becomes increasingly significant as accuracy requirements relax and approximate instead of exact nearest neighbors are found. However, this compromise is negligible, since it was experimentally shown that a speedup of two orders of magnitude was achieved even with 90% accuracy. This percentage is the accuracy of approximation and is defined as the percentage of query points for which the correct nearest neighbor is found.

Among others, it was shown that using as few as seven iterations in the hierarchical k-means algorithm, more than 90% of the nearest neighbor search time efficiency was attained compared to the tree constructed until convergence. Moreover, only 10% of the original build time was required. Concerning k-d trees, speedup over linear search increases as more randomized trees are added to the forest, depending on the dataset. In the case of a 100K SIFT features dataset, it was shown that peak performance was reached with 20 random trees. The researchers also raised the point of dimensionality of the feature space and distribution of the feature points. Given synthetic data, consisting of random points generated from a uniform distribution, speedup decreases dramatically as dimensionality of the points increases. However, in the real world, dimensions are correlated depending on the dataset, often leading even to an increasing speedup as dimensionality increases.

2.4 Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) [49, 48, 34, 43] is a method for creating compact binary descriptors for querying a large database. LSH finds nearest neighbors of points lying in a high dimensional Euclidean space \( \mathbb{R}^n \) in constant time by computing a hash function for a point and rounding a number of random projections of that point into \( \mathbb{R}^1 \). Each random projection contributes a few bits to the descriptor of a point depending on the rounding function. It has been shown in [15] that points that are close together in \( \mathbb{R}^n \) will have similar hash functions; this property can be used to efficiently find approximate nearest neighbors.

2.5 Soft-Assignment

In the bag-of-visual-words model, two image features are considered identical if they are assigned to the same cluster center, but they are considered totally different if they map to different clusters. In other words, the distance between two features is infinite if they are assigned to the same cluster, and zero otherwise. However, it is possible that descriptors corresponding to the same keypoints in different images do not lie close together. This is due to a variety of factors: image noise, varying illumination, instability in the feature detection process, image distortion, etc. can change the descriptor vector. In [45], Philbin et al. introduce an approach called soft-assignment, where a high dimensional descriptor is mapped to a weighted combination of visual words, rather than “hard-assigned” to a single word. The weight assigned to neighboring clusters depends on the distance between the descriptor and the cluster centers. It was shown that soft-assignment can boost the recall of a retrieval system and, if combined with spatial verification, precision can be increased too. However, this technique requires more storage for the index: the authors indexed a dataset
of 5K 1024x768 images with a 1M vocabulary, which resulted in a 36MB index with hard-assignment. Using soft-assignment to the 3 nearest neighbors, the size of the index increased to 108MB.

In Figure 2.3, points A-E represent cluster centers (visual words) and points 1-4 are features. In soft-assignment, features 3 and 4 will be assigned to A, B and C with certain weights, giving more weight to close matches and less to the further. In hard-assignment, features 1-3 are assigned to word A equally, without providing a way of distinguishing that 2 and 3 are closer than 1 and 3.

![Figure 2.3 with soft-assignment, a descriptor can be assigned to a combination of visual words (image appeared in [45])](image)

### 2.6 Query Expansion

In text retrieval, a standard method that improves performance is query expansion, a form of pseudo relevance feedback where a number of the top documents in the list of results are used to reformulate the original query. In particular, a statistical analysis is performed on the terms of the n top documents retrieved. Then, the terms with the highest values of co-occurrence and/or proximity among these documents are added to the original query. In [20], query expansion was introduced to the domain of image retrieval. In a naive application of query expansion as is used in text retrieval, the top-n results from the initial query $Q_0$ are taken and their term-frequency (TF) vectors are averaged and issued to the retrieval system as a new query $Q_1$. The results of the new query are appended to the top-n results of $Q_0$.

There are various methods for expanding a user query, such as transitive closure expansion, recursive average query expansion and multiple image resolution expansion [20]. Chum et al. state that the simplest well performing method is the average query expansion: a new query is formulated by averaging the top-n, $n < 50$ spatially verified results of the original query. A new query $Q_{avg}$ is then formed by the average of the original query $Q_0$ and the n results:

$$d_{avg} = \frac{1}{n+1} (d_0 + \sum_{i=1}^{n} d_i), \quad (1)$$
where \( d_0 \) is the normalized TF vector of the query image region, and \( d_i \) is the normalized TF vector of the \( i \)-th result. The system is queried once more, and the results of query \( Q_{avg} \) are appended to the top-\( n \) results of \( Q_0 \).

### 2.7 Binary Codes

Torralba et al. introduce an attempt to develop efficient and fast image search and scene matching techniques that require little memory, allowing their use on standard hardware and handheld devices. To describe an image, binary codes [54] are created using machine learning techniques. The 384-D Gist descriptor [40] is reduced to a few hundred bits using approaches including boosting, locality sensitive hashing and restricted Boltzmann machine [26], providing a compact bit representation to describe an image in the dataset. In this way, a query can be performed by calculating the Hamming distance\(^1\) of the codes in the database to the code of the query image. Their work is inspired by Salakhutdinov and Hinton, who apply learning techniques and train compact binary codes to perform document retrieval [46].

With as few as 256 bits for each code, a database of 12.9 million images took up less than 600MB of memory, whereas whole Gist descriptors could not fit in main memory, when used in k-d trees. However, even if the Gist vectors had fit, search would have been slow due to the high dimensional space. It was also shown that fast querying is possible: even exhaustive search for exact nearest neighbors from a dataset of 12 million images can be performed in a fraction of a second on a fast computer. Retrieving 1000 nearest neighbors from 20K images using a k-d tree took 3ms/image and retrieved 17% of true neighbors, while binary codes took 6\( \mu \)s/image and retrieved 48% of true neighbors (on the other hand, a forest of randomized k-d trees improves performance, as stated in 2.3).

### 2.8 Product Quantization

In [27], a product quantization approach for approximate nearest neighbor search is introduced, which decomposes high dimensional space into a Cartesian product of low dimensional subspaces and quantizes each subspace separately. Given a high dimensional vector, for example the SIFT descriptor, and a number \( m \), components can be quantized in groups. The input vector \( x \) is split into \( m \) subvectors \( u_j, 1 \leq j \leq m \), of dimension \( D^* = D/m \), where \( D \) is a multiple of \( m \). Each subvector is quantized separately using one of \( m \) distinct quantizers:

\[
\begin{align*}
  u_1(x) = \begin{cases} 
    x_1, & \ldots, x_{D^*+1}, \\
    \vdots, & \ldots, \vdots, \\
    x_{D^*}, & \ldots, \vdots, \\
    \vdots, & \ldots, x_D 
  \end{cases} \rightarrow q_1(u_1(x)), \ldots, q_m(u_m(x)), \quad (2)
\end{align*}
\]

where \( q_j \) is a quantizer associated with the \( j \)-th subvector \( u_j \) and a codebook \( C_j \), which is the set of centroids generated by the quantizer. The codebook \( C \) is defined as the Cartesian product

\[\text{Cartesian product} = \prod_{j=1}^m C_j\]

\(^1\) The Hamming distance between two strings of equal length is the minimum number of substitutions required to change one string into the other.
where the concatenation of centroids of the \( m \) subquantizers define a centroid of \( C \). This approach constructs short codes that describe the feature vectors using a coarse quantizer \( q_c \) and a product quantizer \( q_p \). The coarse quantizer, for example Lloyd’s k-means, associates a vector \( y \) with a centroid \( q_c(y) \), providing the most significant bits in the code, and the product quantizer encodes the residual vector

\[
    r(y) = y - q_c(y), \tag{4}
\]

providing the least significant bits. This procedure is performed only for the feature vectors in the database. A query vector is not quantized and distances are estimated using vector-to-centroid distances, which are explained below.

To approximate the Euclidean distance between a query vector \( x \) and a database vector \( y \), Jégou et al. propose a symmetric and an asymmetric method, as illustrated in Figure 2.4

\[\text{Figure 2.4 Symmetric (SDC) and Asymmetric (ADC) distance computation between two vectors (image appeared in [27])}\]

Symmetric distance computation (SDC): both vectors \( x \) and \( y \) are represented by their respective centroids \( q(x) \) and \( q(y) \). Distance \( d(x, y) \) is approximated by the distance \( \hat{d}(x, y) \triangleq d(q(x), q(y)) \), which is calculated as follows:

\[
    d(q(x), q(y)) = \sqrt{\sum_j d_j(q_j(x), q_j(y))^2}. \tag{5}
\]

Distances are read from a look-up table associated with the \( j \)-th subquantizer, containing pairwise squared distances between centroids of the quantizer.

Asymmetric distance computation (ADC): the database vector \( y \) is represented by \( q(y) \), but the query vector is not quantized. Distance \( d(x, y) \) is approximated by the distance \( \hat{d}(x, y) \triangleq d(x, q(y)) \), which is calculated as:

\[
    d(q(x), q(y)) = \sqrt{\sum_j d_j(u_j(x), q_j(u_j(y)))^2}. \tag{6}
\]
To avoid exhaustive search of approximate nearest neighbors, Jégou et al. use an inverted file, which is built using the coarse quantizer $q_c$. As shown in Figure 2.5, it consists of an array of lists $L_1, \ldots, L_k$, and each list $L_i$ is associated with the centroid $c_i$ of $q_c$ and stores the set $\{ y \in Y: q_c(y) = c_i \}$, where $Y$ is the vector dataset to index. An entry in the list $L_i$, corresponding to $y$, contains a vector identifier and the encoded residual $q_p(r(y))$ (see (4)).

When searching for the nearest neighbors of a query vector $x$, distances are estimated only for a subset of $Y$: only the inverted list $L_i$ corresponding to $q_c(x)$ is scanned. However, $x$ and its nearest neighbors are often quantized to different centroids. For this reason, $x$ is assigned to $w$ indices instead of one, which correspond to the $w$ nearest neighbors of $x$ in the codebook of $q_c$. This approach is called multiple assignment [28], which is similar to the soft-assignment we described in 2.5.

![Figure 2.5: insertion and search procedures in the inverted file (image appeared in [27])](image.png)
2.9 Efficient Similarity Search with Graphs

It is common for users to associate their shared images with landmarks and events, such as popular historical monuments or music concerts. The rising popularity of photo sharing services renders impractical for a user to browse through such huge volumes of pictorial content, even with the use of tags.

Papadopoulos et al. propose a novel scheme for automatic detection of landmarks and events in annotated collections of images by means of image similarity graphs and exploiting both visual and tag similarity [42]. For every image in the collection, 128-dimensional SIFT descriptors are computed, and the bag-of-visual words model is applied with the use of a vocabulary of 500 words and a k-means clustering process. An image feature is assigned a visual word using the code word uncertainty model [55]. A similarity function computes the pairwise similarity between images and the most similar images to each image are represented as neighboring nodes in the graph. In this way, a visual similarity graph is constructed. A second graph, based on tag similarity is created based on the co-occurrences of tags in the image metadata. An inverted index maintains a list of annotated images for each tag. For each pair of images in this list, an undirected weighted edge is created linking the two images in the graph. As shown in Figure 2.6, the two graphs are merged together and a community detection [47] based clustering is performed on the resulting hybrid image graph, which locates groups of nodes that are more densely connected to each other than the rest of the nodes. Finally, the clusters found are classified as either landmarks or events using kNN and Support Vector Machine classification techniques.

![Figure 2.6: distinguishing between landmarks and events with a hybrid image similarity graph (image appeared in [42])](image-url)
Jing and Baluja address the image-ranking problem by identifying authority nodes in a visual similarity graph [29] (see Figure 2.7) inspired by the PageRank [41] algorithm. As in PageRank, the graph measures the importance of each node as well as the probability that a user visits node \( j \) after visiting node \( i \): if an image \( i \) is related with a visual hyperlink to an image \( j \), then it is probable that a user visits \( j \) after viewing \( i \). Again, similarity between the SIFT descriptors of two images determines the connection between nodes in the graph, and it is defined as the number of points of interest the two images have in common over their average number of points of interest. The ordering of images in the list of results is determined by the centrality of their corresponding nodes, which is a measure of importance.

*Figure 2.7: similarity graph generated from the top-1000 search results of "Mona-Lisa". The largest two images are the most important, as they contain the highest rank (image appeared in [29]).*
3 Building a More Distinctive Visual Vocabulary

To build the Visual Vocabulary, the first step is the extraction of the feature set from the image collection. Quantization of the feature set will build the initial visual words of the vocabulary, which we enrich with the application of a new method explained later in this section.

3.1 The Feature Set

3.1.1 Overview of SIFT Keypoints and Descriptors

In the context of our application, a feature denotes a piece of information extracted from points, edges, corners or even complex structures like objects in a specified image. We used the SIFT algorithm [33], which extracts descriptor vectors from the points of interest (keypoints), after detecting them using the Harris-Laplace [24] detector. Other approaches detect features using affine-invariant Hessian regions [36] or dense sampling [23]. A SIFT-keypoint (see Figure 3.1(a)) is a circular image region with an orientation, described by four parameters: the keypoint center \( x \) and \( y \), scale (radius) and orientation in radians. Because SIFT detects keypoints at multiple scales and positions, it introduces scale, rotation and translation invariance.

![Figure 3.1](http://www.vlfeat.org/api/sift.html)

Figure 3.1 a SIFT keypoint is characterized by its location, scale and orientation

As mentioned in section 1, a feature vector describes the appearance of the point of interest in such manner that similar vectors belong to similar points of interest. Similarity can be calculated using similarity or distance measures like cosine similarity or L2 distance, respectively.

Technically, a SIFT descriptor is a 3-D spatial histogram of the image gradients. A 16x16 window, broken into sixteen 4x4 windows is taken around a keypoint (see Figure 3.2). Within each 4x4 window, gradient magnitudes and orientations are calculated, and the latter are put into an 8 bin histogram. Gradient orientations in the range 0-44 degrees are added to the first bin, 45-89 are added to the next bin, and so on. The amount added to the

---

2 Image appeared in http://www.vlfeat.org/api/sift.html
histogram depends on the gradient magnitude and its distance from the keypoint: the further the gradients are from the keypoint, the smaller are the values that will be added to the histogram. Performing the above for a 4x4 window results in 8 numbers, therefore all 16 4x4 windows generate 16x8 = 128 numbers, hence the dimensionality of the SIFT feature vector. Figure 3.1(b) shows matched features of keypoints between an object and an image in which the object appears.

![Figure 3.2](image)

**Figure 3.2:** for each of the 16 4x4 windows, 16 gradient orientations are put into an 8-bin histogram (image appeared in http://www.aishack.in/2010/05/sift-scale-invariant-feature-transform/)

Given a collection of images, we execute a batch process that extracts 128-dimensional vectors using simple SIFT\(^4\) and stores them in the file system. Features are saved in a two-file scheme, one consisting of the actual vectors and another pointing each filename to the range of its respective vectors, as depicted in Figure 3.3. \(Img_x\) is the absolute path of image \(x\), which is described by a feature set \(F_x = \{\hat{f}_{x,1}, \hat{f}_{x,2}, \ldots, \hat{f}_{x,|F_x|}\}\). To save or load the feature set, a process similar to Code 1 is executed.

![Figure 3.3](image)

**Figure 3.3:** scheme for storing the feature sets of a collection of images

\(^4\) Extraction of SIFT features from images was performed with ColorDescriptor software [55, 13].
3.2 Quantization of the Feature Set

Having stored the features of the collection, quantization of the feature space can take place, in order to produce the initial visual vocabulary. At a later step, this vocabulary will be enriched with composite visual words to improve retrieval.

3.2.1 Clustering Large Volumes of Data

Quantization can be performed by applying a clustering algorithm on the extracted feature vectors. We used Lloyd’s k-means included in VLFeat [3], an open source library of popular computer vision algorithms. To cluster the data, they must be provided as input for the clustering algorithm, i.e. they have to be loaded in main memory, which is impossible in the case of millions of multidimensional features. Even simple k-means clustering without triangular inequality checks [22] cannot run in systems with limited main memory capacity and such large volumes of data, since it would require $\pi d$ space, where $\pi$ is the number of points and $d$ their dimensionality.

To this end, we partition the input vectors and provide each partition to a streaming implementation of the k-means algorithm [14]. As illustrated in Figure 3.4, the set $S$ of the extracted feature vectors is divided to $S_i$ subsets and k-means is applied for each subset. The union of the resulting cluster centers is then provided to a second execution of the algorithm, producing the final centers. Code 2 shows our implementation of this technique.

The user provides the feature set (file A in Code 1), the desired size of each partition in rows and the number of centers. For each partition $S_i$, a k-means++ process precedes k-means, in order to find appropriate initial centers to avoid an arbitrarily poor approximate solution of k-means (this issue will be explained in 3.2.2). Clustering each $S_i$ produces a center set $T_i$, which we save in a file. Finally, k-means is applied to $\bigcup_i T_i$, producing the final centers $C$. Center set $C$ is our initial vocabulary which we enrich at a later point to improve retrieval.

---

5 K-means with triangular equality checks requires $\frac{c(c-1)}{2}$ extra space to store the distances between centers, where $c$ denotes the number of centers.
Algorithm: Streaming Divide and Conquer K-means
//vectors, partition size in rows, number of centers
Input: vectors V, integer p, c;
Output: centers C;

integer loops = \left\lfloor \frac{\text{V.rows}}{p} \right\rfloor ;
integer rows_per_loop = \left\lfloor \frac{\text{V.rows}}{\text{loops}} \right\rfloor ;

for (integer i=0; i<\text{loops}; i++)
    S_i = \text{loadPartition}(V, \text{rows_per_loop});
    \text{doKmeansPlusPlus}(S_i, c);
    T_i = \text{doKmeans}(S_i, c);
    \text{saveCenters}(T_i);

\text{doKmeansPlusPlus}(\bigcup_i T_i, c);
C = \text{doKmeans}(\bigcup_i T_i, c);
\text{return } C;

Code 2: our implementation of the streaming divide and conquer for k-means

3.2.2 Avoiding a Poor Clustering with K-means++

Lloyd’s k-means finds an approximate solution that minimizes the sum of squared distances from each data point to its cluster center. In particular, for an integer k and a set of data points \(X\) of \(d\) dimensions, \(X \subseteq \mathbb{R}^d\) we wish to choose a set of \(k\) centers \(C\) to minimize the potential function:

\[
\varphi = \sum_{x \in X} \min_{c \in C} \|x - c\|^2. \tag{7}
\]
However, this algorithm can suffer from a serious flaw: this approximation can be arbitrarily bad compared to optimal clustering.

![Figure 3.5: a suboptimal k-means clustering](image)

Suppose the data points $p_1$, $p_2$, $p_3$, $p_4$ form a rectangle with larger width than height, and our initial cluster centers $c_1$ and $c_2$ lie in the middle of the upper and lower side of the rectangle respectively, as shown in Figure 3.5. K-means will converge without relocating the two centers, forming two clusters: one containing points $p_1$ and $p_3$, and another containing the points at the lower corners of the rectangle, $p_2$ and $p_4$. Because the height of the rectangle is smaller than its width, this is a suboptimal clustering according to (7).

![Figure 3.6: k-means clustering can be arbitrarily suboptimal](image)

Horizontally stretching the rectangle to an arbitrary width (see Figure 3.6) the standard k-means algorithm will still perform suboptimally. Increasing the horizontal distance between data points $p_1$, $p_2$ and $p_3$, $p_4$ induces an arbitrarily poor clustering.

To address this issue, we employ an implementation [3] of the k-means++ algorithm [16] prior to k-means, to avoid selecting random data points as initial centers. Let $X$ denote the data points, $k$ the desired number of clusters and $D(x)$ the shortest distance from a data point to the nearest cluster center already chosen. The k-means++ algorithm chooses the initial centers as follows:
1. Pick a center \( c_1 \) uniformly at random from \( X \).
2. For each data point \( x \), compute \( D(x) \).
3. Pick a data point \( x \in X \) with probability \( \frac{D(x)^2}{\sum_{x \in X} D(x)^2} \), as the new center \( c_i \).
4. Repeat steps 2 and 3 until \( k \) centers are picked.
5. Proceed with standard \( k \)-means clustering with the initial centers chosen.

Although the selection of initial centers consumes extra time, the following \( k \)-means converges much faster, i.e. less iterations are executed. However, the main benefit lies in good seeding; we will enrich our initial vocabulary with composite visual words formed by initial visual words, therefore a poor clustering must be avoided. Approximation ratio \([14]\) quantifies this benefit: let \( \varphi_C \) be the potential of the clustering \( C \) returned by the algorithm (see (7)), and \( \varphi_{C^{OPT}} \) be the potential of the optimal clustering \( C_{OPT} \). Approximation ratio is defined as the worst case ratio \( \frac{\varphi_C}{\varphi_{C^{OPT}}} \).

\( k \)-means++ guarantees an approximation ratio \( O(log k) \) for \( k \) clusters, whereas standard \( k \)-means can generate arbitrarily worse clusterings than the optimum \( C_{OPT} \).

### 3.2.3 Lowering the Time Required for Quantization

During the quantization step, Lloyd’s \( k \)-means algorithm introduces significant time (and space) complexity requirements due to the large number of vectors to cluster and cluster centers to calculate. The simple \( k \)-means (Lloyd’s algorithm) has \( O(n d c l) \) complexity, where \( n \) is the number of vectors to cluster, \( d \) the dimensionality of the vectors, \( c \) the number of centers to find and \( I \) the number of iterations of the algorithm.

The above fact renders the application of \( k \)-means impractical in this context, since millions of features are extracted and the number of cluster centers to find can reach up to one million, in order to build a distinctive visual vocabulary. This problem can be addressed at indexing time with the application of a technique which does not require many centers in order to assign a visual word to a feature vector. The essence of this technique—which is our main contribution—lies in composing additional visual words: for each of the feature vectors, the top-\( k \) visual words (cluster centers) closest to it are found and their labels are concatenated. Thus, a composite visual word is formed which describes the vector with more than one initial words. Therefore, the previous clustering step can be executed with extremely less centers (even as few as 50) reducing time requirements, and at the same time providing satisfactory query results. We elaborate on this technique in the following subsection.

### 3.3 Indexing the Collection with Composite Visual Words

#### 3.3.1 The Bag-of-visual-words Model

After quantization, a set \( C \) of cluster centers is created. Then, each feature vector is compared to the cluster centers in \( C \) according to their L2 distance:

\[
D(x, c) = \sqrt{\sum_{i=1}^{n} (x_i - c_i)^2}, \tag{8}
\]
where \( x \) denotes the feature vector, \( c \) a cluster center and in this case, \( n = 128 \), since SIFT extracts 128-dimensional features. In order to index the collection according to the bag-of-visual-words model, each vector is assigned the label of the nearest cluster center:

\[
Label_x = \arg \min_{c \in C} D(x, c). \quad (9)
\]

For each image in the collection, a document is created, which contains the labels of the centers that are closest to its features, as shown in Figure 3.7. Eventually, each document is analyzed and indexed with methods deployed in text retrieval, building an inverted index (see Figure 3.8) of terms pointing to documents. A typical inverted index consists of an array of the vocabulary terms, and each entry in the array contains a list of the documents in which the term appears. The importance of each term \( t \) is determined by TF-IDF weighting, namely its term frequency (TF), and its inverse document frequency (IDF). Term frequency \( TF(t, d) \) measures the number of appearances of term \( t \) in a document \( d \), and it is usually normalized by the number of appearances of the most frequent term in \( d \). Inverse document frequency measures the general importance of \( t \) in the collection of documents:

\[
IDF(t) = \log \frac{N}{DF(t)}. \quad (10)
\]

where \( N \) is the number of documents in the collection and \( DF(t) \) stands for the number of documents in which \( t \) appears. Then TF-IDF is calculated as:

\[
TF - IDF(d, t) = TF(t, d) \times IDF(t). \quad (11)
\]

An important term has a high \( TF \) and a low \( DF \). In other words, it must appear often in a certain document \( d \) but rarely in the whole collection.

\[\text{Figure 3.7: In the bag-of-visual-words model, for each feature of an image, the label of closest cluster center is kept, forming a document of visual words which will be indexed as in text retrieval}\]
3.3.2 Composite Visual Words

As mentioned, an indexed visual word is the label of the cluster center closest to a feature vector of an image. If the initial visual vocabulary is limited, (perhaps because the clustering process was impractical for thousands of centers) then the resulting visual words will not be distinctive enough, thus inflicting a decrease on the precision and recall of the retrieval system. For each feature vector, instead of indexing the label of its nearest center only, we index the concatenation of the labels of the nearest $k$ centers in ascending order of their L2 distances from the current feature vector.

Figure 3.9 depicts the above idea: the composite visual word "ACB" corresponds to a feature vector lying nearest to center $A$, then $C$ and $B$, belonging to the space where points lie closer to $A$, then $C$ and $B$. Similarly, visual word "BCA" is created by a feature vector that lies nearest to $B$, $C$ and $A$, in ascending order of distance.

This approach implicitly exploits the inter-center relationships around neighboring clusters so that the resulting composite visual word, as a permutation of relevant visual words,
better describes its corresponding descriptor, even with a small number of centers. In our experiments, as few as 50 centers can offer up to 0.38 mean average precision.

Explaining our method from the aspect of the Voronoi diagrams, this kind of diagram contains cells of points whose distance from the center of the cell is not greater than their distance from the other centers. Figure 3.10 illustrates a Voronoi diagram in 2-dimensional space. Extending this rationale, assigning composite visual words to feature points defines a cell that:

a. contains points whose distance from the $k$ nearest centers (e.g. cluster centers $X$, $Y$ and $Z$, if $k = 3$) is not greater than their distances from the other centers and

b. the ranking of the $k$ centers nearest to the points is the same, according to the ascending order of distance.

Figure 3.10: a Voronoi diagram in 2D space

Making composite visual words is very similar to soft-assignment [45]; however, there is a key difference between the two techniques: in soft-assignment, a feature is assigned to several visual words (cluster centers) separately, whereas in making composite visual words a feature is assigned to their concatenation, thus enriching the resulting visual vocabulary.

At this point, we make a distinction between the set of centers $C$ and their labels: the labels of the cluster centers in $C$ form the initial visual vocabulary $V$. Since a composite word $w'$ is formed by a permutation of words in $V$, this method creates an extended vocabulary $V'$ from the initial vocabulary $V$. The maximum number of possible words in $V'$ is:

$$|V'|_{max} = P(|V|, k), \quad (12)$$

where $k$ is the user defined number of words in $V$ that will form a composite word, and $P(|V|, k)$ the $k$-permutations of $|V|:

$$P(|V|, k) = \frac{|V|!}{(|V| - k)!}, \quad (13)$$

For example, if $|V| = 100$ and $k = 3$, then the maximum number of words that $V'$ can possibly contain is $100 \cdot 99 \cdot 98 = 970,200$. For $k = 4$, then $|V'| = 94,109,400$. Of course, a vocabulary of such size would have a negative impact on retrieval, because a lot of terms would appear only once in the collection. For this reason, we apply thresholds on the distance of features from candidate centers, as explained in 3.3.3.
3.3.3 Distance Thresholds

To refrain from selecting the label of the nearest, nevertheless remote center to represent a feature vector, we apply a threshold on the distance of each center from a feature vector. If their distance is above this threshold, then the center is disregarded and no assignment is made. In this way, we reduce the possibility of assigning a feature vector to a visual word that does not characterize it; therefore we build a more distinctive vocabulary.

A restriction on distance would be to disqualify a center \( c \) if:

\[
D(c, x) > a \cdot \max_{c' \in C} D(c', x), \tag{14}
\]

where \( a \in (0,1) \) and \( \max_{c' \in C} D(c', x) \) is the maximum distance between feature \( x \) and a center.

We extend the above condition in order to include \( k \) nearest candidate centers, instead of the single nearest. Instead of a fixed condition for all top-\( k \) centers, we introduce a constraint that becomes progressively stricter for the next nearest center. The \( i \)-th nearest center \( c \), \( 1 \leq i \leq k \), is disqualified if:

\[
D(c, x) > e^{-\alpha i} \max_{c' \in C} D(c', x), \tag{15}
\]

where \( a \) and \( \max_{c' \in C} D(c', x) \) are the same as in (14). Increasing \( a \) makes the condition stricter. In this case, notice that we do not have to set a fixed \( k \) explicitly; a composite visual word may consist of many labels as long as their respective centers satisfy the above condition. On the other hand, given a distance threshold and a user defined number of words \( k \) that will form a composite word, it is expected that there will be composite words formed by less than \( k \) words. Constants \( a, k \) and \( |V| \) strongly affect the size of the composite vocabulary \( V' \). Now, the maximum number of words in \( V' \) is:

\[
|V'|_{\max} = \sum_{i=1}^{k} P(|V|, i). \tag{16}
\]

which is greater than (12), which means that it is possible to build an even worse vocabulary \( V' \). However, this is highly unlikely if a good threshold is set, as shown in section 4. Apart from that, collections usually contain images that share a lot in common; therefore the resulting vocabularies do not contain too many unique words.
Algorithm get Composite Visual Word

//the feature vector and a list of cluster centers
Input: vector \( x \), centers \( C \);
float \( a \), integer \( k \);
Output: string \( w \); //the composite visual word

//compute the L2 distance of each center from the vector
for each \( c \in C \)
\[
c.\text{distance} = \text{computeDistance}(c, x);
\]

\text{sort}(C); //sort \( C \) in ascending order of distance from \( x \);
float \( d_{\text{max}} \) = \text{C.last().distance};
integer \( i = 1 \);
string \( w \);

\text{while} (i \leq k) //for the \( k \) nearest centers
\[
\text{float } d_i = \text{C.elementAt(i).distance};
\text{float threshold } = e^{-a}d_{\text{max}};
\]
//concatenate the label of the next nearest center
if \( d_i \leq \text{threshold} \) \( w = w.\text{concat}(\text{C.elementAt(i).label}); \)
else \text{break};
\( i++ \);
return \( w \);

Code 3: Algorithm that forms composite visual words

Code 3 shows the process that creates a composite visual word given a feature vector of an image. At first, it calculates the distances of the cluster centers from the given vector and sorts them in ascending order. Then it concatenates the labels of the nearest \( k \) centers, as long as their distances are less or equal to the threshold. If a center \( c_i, 1 \leq i < k \) is disqualified, the process does not continue for \( c_{i+1} \); it would not satisfy the condition, as \( D(c_{i+1}, x) \geq D(c_i, x) \), since centers are sorted.

3.3.4 Indexing with Apache Solr

Apache Solr [4] is an open-source search platform that extends the Lucene [5] search library. Lucene is an open-source library for information retrieval software, while Solr provides various features such as faceted search and filtering, advanced text analysis, an administrator interface, rich document parsing and indexing, etc.

The above process (see Code 3) is performed for the whole feature set of the collection of images. For each image, a document is created (see Figure 3.7) that holds the composite words found, along with a unique id for the image. This document is then forwarded to a local Apache Solr server for indexing.
Figure 3.11 shows a document retrieved after indexing the collection with $k = 3$ and $\alpha = 0.2$, from an initial vocabulary $V$, $|V| = 50$. The visual words are in the description tag. Labels of cluster centers are in the form “cc_x”. For example “cc_13” is not a composite word, since it has no further concatenations of labels. Obviously, the feature vector which results in this word does not lie near any other center close enough to satisfy the increasingly restricting distance constraints. On the other hand, “cc_4cc_42” is a composite word consisting of “cc_4” and “cc_42”. In this case, the feature vector related to it lies near two centers that meet the distance requirements.

3.4 Query Processing

After building the index, retrieval can take place with user provided queries. A query is a user provided image forwarded to the retrieval system in order to produce relevant results which are then returned to the user in a ranked list, ordered by relevance of each result to the query. As mentioned in section 3.3.1, retrieval is reduced to a text IR problem, since the visual words of images are represented by the labels of the cluster centers closest to its feature vectors.

As a prerequisite step, in order to search the inverted index for relevant images, the feature vectors of the query image are extracted. Each feature vector is replaced by the concatenated labels of its closest $k$ cluster centers according to their Euclidean distance and a distance threshold, as described in section 3.3.3.

Having replaced the query image by a bag of words, the resulting text query is sent over to the retrieval system. In our case, the query document is sent via HTTP POST request method to the Solr server and evaluated according to the vector space model. Typically, the query – as well as every indexed document- is represented by a vector in the space of vocabulary terms with TF-IDF weighting, and it is compared with the documents that have at least one term in common. As mentioned in 3.3.1, the inverted index contains an array of terms in the vocabulary, and each entry contains a list of documents in which the term appears (see Figure 3.8). In order to compare the query vector to the document vectors, only the lists of the terms contained in the query have to be retrieved, thus avoiding linear search through the whole collection. Lucene employs cosine similarity to measure the similarity between vectors:
\[ \text{sim}(q, d) = \frac{q \cdot d}{\|q\| \|d\|} = \frac{\sum_{i=1}^{n} q_i \times d_i}{\sqrt{\sum_{i=1}^{n} (q_i)^2 \times \sum_{i=1}^{n} (d_i)^2}} \]  

which is calculated as the dot product of the query and document vectors over their magnitude. We provide additional functionality which allows the user to specify a term frequency threshold to rule out non-distinctive visual words, before submitting a query.

### 3.5 User Interface

In order to accommodate an automated and easy to use environment for our tests and evaluation, a graphical user interface was implemented (in Java) along with core functionality. Although the core already provides an API that can be used with simple calls, the GUI provides an additional abstraction which favors future tests, especially for third parties.

![Figure 3.12: the main window of the user interface](image)

Figure 3.12 shows the main window of the application, where the user can initiate the core functions. As illustrated, a user may:

- Extract the features of an image collection in a specified directory,
- index the collection with the bag-of-visual-words-model at the Solr server and
- evaluate the quality of the retrieval by measuring the mean average precision of issued queries, based on ground truth.

At first, some parameters must be configured, as depicted in Figure 3.13. The user specifies the paths necessary for the above functionality, such as the location of the collection, the file where the features are saved, the address of the Solr server, etc. Among others, parameters \( k \) and \( a \) can be set (see 3.3.3), as well as frequency thresholds for the query terms. It should also be noted that our application not only computes the SIFT descriptor, but also 64-dimensional SURF [18] using the corresponding API functions of the OpenCV library of computer vision [6]. Quantization of the feature space is performed by a separate executable file, implemented in C using VLFeat, as mentioned in 3.2.1.
Figure 3.13: the configuration window of the user interface
4 Experimental Evaluation

Image and object retrieval systems can be evaluated with datasets of images. These datasets are collections of images publicly available on the web, such as the MirFlickr Retrieval Evaluation [7], ImageNet [8], the NUS-WIDE image database [9], the Visual Object Classes Challenge [10], and so on. We used the Oxford Buildings [1] and the Paris Buildings [2] datasets, comprising images from Flickr [11] by searching for particular Oxford and Paris landmarks, respectively. Preceding the evaluation, features are extracted from the images and quantized. Every image is represented by a document of visual words, which is then passed for indexing.

Evaluation is carried out by submitting queries, retrieving results and applying known performance measures for retrieval systems. To use such measures, it is necessary to distinguish between relevant and non-relevant results for each query. This knowledge, known as ground truth, was provided for a set of 55 queries on the Oxford and Paris datasets.

4.1 Performance Measures

Precision and Recall are two fundamental measures of performance in retrieval systems:

$$P = \frac{|\text{Rel} \cap \text{Ret}|}{|\text{Ret}|}, \quad (18)$$

$$R = \frac{|\text{Rel} \cap \text{Ret}|}{|\text{Ret}|}, \quad (19)$$

where $\text{Rel}$ is the set of relevant documents and $\text{Ret}$ the set of retrieved documents given a particular query. Precision is the fraction of retrieved documents that were relevant to the query and Recall is the fraction of relevant documents that were successfully retrieved. Since the order of the returned results is crucial for the retrieval systems, precision and recall values do not suffice. We need to compute precision and recall at every position of the ranked list of results. To achieve this, we apply the Average Precision metric for each query:

$$AP = \sum_{k=1}^{n} P(k)\Delta r(k), \quad (20)$$

where $k$ is the rank in the list of retrieved documents and $n$ is the number of retrieved documents. $P(k)$ denotes the current average precision at the $k$-th and $(k-1)$-th position in the list and $\Delta r(k)$ the difference of recall between the $k$-th and the $(k-1)$-th result. Finally, we calculate Mean Average Precision for the set of queries as a whole:

$$mAP = \frac{\sum_{q \in Q} AP(q)}{|Q|}, \quad (21)$$

where $Q$ is the set of the ground truth queries.
4.2 The Oxford Buildings Dataset

The Oxford buildings dataset comprises 5062 images from Flickr searches for Oxford landmarks (see Figure 4.1 and Table 1). We limited the number of extracted features per image to two thousands. The collection has been manually annotated to generate a ground truth for 5 queries for 11 different landmarks, resulting in 55 different queries for which relevant and non-relevant images are known. Thus, a retrieval system can be evaluated.

Each image in the dataset is characterized by one of four possible labels:

- GOOD: a clear picture of the object/building.
- OK: more than 25% of the object is clearly visible.
- JUNK: less than 25% of the object is visible, or there are very high levels of occlusion or distortion.
- BAD: the object is not present.

The calculation of mean average precision takes into account images that bear the labels “GOOD” and “OK”.

![Figure 4.1: images from the Oxford buildings dataset, each depicting a different landmark](image-url)
Table 1: number of images and SIFT vectors extracted from the Oxford Buildings dataset

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Images</td>
<td>5062</td>
</tr>
<tr>
<td>Features</td>
<td>9,731,989</td>
</tr>
<tr>
<td>Average features per image</td>
<td>1923</td>
</tr>
</tbody>
</table>

### 4.2.1 Mean Average Precision

We calculated the mAP (see (21)) for two values of $k$, where $k$ is the number of nearest centers that form each composite visual word. Obviously, a $k = 1$ setting does not generate any composite words, which allows us to compare the mAP of our proposed method against the standard method. We applied the distance condition (15), described in 3.3.3, keeping a constant $\alpha = 0.2$.

![mAP](image)

**Figure 4.2:** mAP results for increasing initial vocabulary size

Figure 4.2 illustrates the mAP results for increasing size of the initial vocabulary $V$, i.e. increasing number of cluster centers produced after the quantization step (see 3.2). At this point, it should be noted that no supplementary mechanism for enhancing the retrieval was utilized (e.g. soft-assignment [45], query expansion [20] and spatial verification [44]).

For as few as 50 centers, we achieved 0.383 mAP, whereas the baseline reached 0.087 mAP only. In Table 2, we compare our method to three clustering methods applied to a similar bag-of-visual-words model and measured in [44]. A maximum of 0.44 mAP was attained with our method for $|V| = 200$ only, while Philbin et al. attained 0.355 mAP with $|V| = 10K$ and exact nearest neighbors, slightly outperforming our method in the case of $|V| = 1M$, which achieved 0.464 mAP. However, it should be noted that their clustering process used 5M descriptors only. Notice that our method slightly outperforms hierarchical k-means (HKM), which attained 0.439 mAP with as many as 1M visual words generated from 16.7M descriptors.
descriptors. Approximate k-means (AKM) reaches 0.618 mAP with 1M cluster centers using a forest of eight randomized k-d trees. It should also be mentioned that we performed image retrieval on our datasets, i.e. we submitted whole images as queries. On the other hand, Philbin et al. performed object retrieval, which limits the interference surrounding the object of interest.

Performance degrades as $V$ rises, but this is natural since $|V|$ affects the size of the composite vocabulary $|V'|$, as stated in 3.3.3. If $V'$ is too rich then probably different words are assigned to similar feature vectors, inflicting a negative effect on the retrieval. Nevertheless, we still can build an image retrieval system with acceptable performance in less time, because we would have to quantize the feature space among 50 clusters only. In the case of k-means, with $O(ndcl)$ time complexity and $c = 50$, we quantize the feature space 10 times faster than the case of $c = 500$ with little loss of mAP.

<table>
<thead>
<tr>
<th>Approach</th>
<th># of descriptors</th>
<th>Vocabulary size</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-means</td>
<td>5M</td>
<td>50K</td>
<td>0.464</td>
</tr>
<tr>
<td>HKM</td>
<td>16.7M</td>
<td>1M</td>
<td>0.439</td>
</tr>
<tr>
<td>AKM</td>
<td>16.7M</td>
<td>1M</td>
<td>0.618</td>
</tr>
<tr>
<td>Our approach</td>
<td>9.7M</td>
<td>$</td>
<td>V</td>
</tr>
</tbody>
</table>

Table 2: comparison between our method and exact k-means, hierarchical (HKM) and approximate (AKM) k-means, as measured in [44].
Table 3: top and bottom 3 queries for k=3 and |V|=200

<table>
<thead>
<tr>
<th>Top-3 Queries</th>
<th>Bottom-3 Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Query</strong></td>
<td><strong>Average Precision</strong></td>
</tr>
<tr>
<td>pitt_rivers_2</td>
<td>1.0</td>
</tr>
<tr>
<td>pitt_rivers_5</td>
<td>0.8037</td>
</tr>
<tr>
<td>pitt_rivers_3</td>
<td>0.8033</td>
</tr>
</tbody>
</table>

Figure 4.3: top-5 results of the top-3 queries for k=3 and |V|=200 (the first row depicts the query images)
Since with an initial vocabulary of 200 words and \( k = 3 \) we achieved the highest mAP (0.44), we show the top and bottom 3 queries for this setting (see Table 3, Figure 4.3 and Figure 4.4). Figure 4.5 contrasts our method to the standard method with 10-bin histograms of average precision scores, for the set of 55 queries. Without a composite vocabulary, the histogram of scores is skewed to the left, i.e. many queries scored less than 0.5. For example, the setting \( k = 3, |V| = 200 \), which attained the highest mAP, produced a more balanced histogram (see Figure 4.5 (e) and (f)) than the one for \( k = 1, |V| = 200 \). With our technique, 26 queries scored more than 0.5 AP whereas in the standard method, with \( |V| = 200 \), this happened for 13 queries only. However, the standard method surpasses our technique for \( |V| = 750 \). For the \( k = 1 \) setting, 23 queries scored above 0.5, while for the \( k = 3 \) setting this happened for 17 queries only, probably due to a too rich composite vocabulary \( V' \) generated from \( V \) (see Table 4).
Figure 4.5: Histograms of AP score frequencies
4.2.2 Building the Inverted Index

Table 4 shows the number of words in the resulting composite vocabulary $V'$ created from $V$, with $k = 3$ and $a = 0.2$. As stated in 3.3.3, the number of words in $V'$ can be controlled with thresholds and an appropriate value of $a$ and $k$. Here, with 750 words in $V$, $|V'|$ is less than 1M, which is still too large, because it causes mAP to decrease.

| $|V|$ | $|V'|$ |
|------|------|
| 50   | 3,688|
| 100  | 26,978|
| 200  | 120,950|
| 500  | 599,683|
| 750  | 980,644|

Table 4

In Figure 4.6 we can see that the size of Solr’s inverted index is substantially larger in the case of $k = 3$ than $k = 1$. However, the total index size will practically never be too big to fit in main memory, since the number of centers is low and control over the size of the composite vocabulary $V'$ is provided through constants $a$ and $k$.

We also measured the time required to build the inverted index on a single desktop pc\(^6\). As illustrated in Figure 4.7, for up to 200 centers, indexing time for $k = 3$ is comparable to $k = 1$. These results are determined by the algorithm in Code 3, which is implemented in Java. It calls the Collections’ sort from the Java API, which is a modified mergesort that requires $O(n \log n)$ time to sort the distances of $n$ centers from the current vector in ascending order. Apart from that, distances for $n$ centers must be computed preceding the sort. We should also note that no sorting is made for $k = 1$, since only the nearest and most distant center are needed, which can be found while computing the distances. For 500 and

---

\(^{6}\) CPU: Intel Core 2 Duo E6750 @ 2.66 GHz, RAM: 4GB Dual Channel, HDD: external, USB 2.0, 7200rpm, OS: Win7 x64, Solr version: 3.4, Java: JavaSE 1.6
750 centers, time is not linear to the number of centers; however, we are still able to build a retrieval system with acceptable mAP (0.38 for \( k = 3 \) and \( |V| = 50 \)) with ten times faster quantization than the case of \( |V| = 500 \) and \( k = 1 \), and 6.38 times faster indexing (0.99h instead of 6.32).

Let \( QT \) be the time to quantize the feature space and \( IT \) the time to index the collection:

\[
\begin{align*}
\text{Speedup}_{QT} &= \frac{QT_{|V|=500}}{QT_{|V|=50}} = \frac{nd10cI}{ndcI} = 10, \\
\text{Speedup}_{IT} &= \frac{IT_{k=1,|V|=500}}{IT_{k=3,|V|=50}} = \frac{6.32}{0.99} = 6.38.
\end{align*}
\]

Another example that indicates the efficiency gains of our method is the \( |V| = 200, k = 3 \) setting against the \( |V| = 750, k = 1 \) setting, which attains a comparable mAP (0.012 greater in the first case):

\[
\begin{align*}
\text{Speedup}_{QT} &= \frac{QT_{|V|=750}}{QT_{|V|=200}} = \frac{nd3.75cI}{ndcI} = 3.75, \\
\text{Speedup}_{IT} &= \frac{IT_{k=1,|V|=750}}{IT_{k=3,|V|=200}} = \frac{12}{3.6} \approx 3.33.
\end{align*}
\]

Figure 4.7: indexing time for increasing initial vocabulary \(|V|\)
4.2.3 Query Processing

A query submission to our retrieval system consists of three different steps (see 3.4): (a) extraction of SIFT features limited to 2,000 from the query image, (b) generation of a document of visual words from the features and (c) forwarding the query document to the Solr server for evaluation and retrieval.

![Average Query Processing Time](image)

**Figure 4.8:** average time required to process each of the 55 queries

About 20 seconds were consumed for the first step of the above process, i.e. the extraction of SIFT features from the query image. We believe this is natural, since the extraction was performed by a software implementation, with the processing power of a desktop pc. Of course, there are more efficient approaches for computing the local descriptor of an image, such as SURF. This process can be accelerated if executed on a server, with the assist of graphics processing units [25] and CUDA technology [58]. Field Programmable Gate Arrays (FPGAs) [59, 19, 53] have also been employed in hardware implementations of feature detection.

A query to the Solr server takes ~1 sec, and the rest of the time is spent creating the document of visual words, calling an implementation of the algorithm in Code 3. As illustrated in Figure 4.8, for up to 200 centers, time is comparable between the two methods, rendering the $|V| = 200$, $k = 3$ configuration the most appropriate in respect to mAP performance and query execution speed.
4.3 The Paris Buildings Dataset
The Paris buildings dataset comprises 6412 images from Flickr searches for landmarks in Paris (see Figure 4.9 and Table 5). In this case, we limited the number of extracted features per image to one thousand. Similarly to the Oxford Buildings dataset, this collection has been manually annotated to generate a ground truth for 55 different queries. Again, each image in the dataset is classified with one of four possible labels: GOOD, OK, JUNK or BAD.

Figure 4.9: images from the Paris buildings dataset, each depicting a different landmark

<table>
<thead>
<tr>
<th>Images</th>
<th>6412</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features</td>
<td>6,314,776</td>
</tr>
<tr>
<td>Average features per image</td>
<td>985</td>
</tr>
</tbody>
</table>

Table 5: number of images and SIFT vectors extracted from the Paris Buildings dataset
4.3.1 Mean Average Precision

![Figure 4.10: mAP results for increasing vocabulary size](image)

We compared the $k = 1$ to the $k = 3$ setting using a constant $\alpha = 0.2$ and the distance condition (15). There is still a considerable increase in mAP with our method, as illustrated in Figure 4.10, with a small decrease for 750 centers, as the standard method tends to outperform the k=3 setting for larger initial vocabularies. As mentioned in 4.2, a too rich vocabulary contains different words that describe similar features. Since the size of the composite vocabulary $V'$ depends on that of the initial vocabulary $V$ and constants $\alpha$ and $k$, $V'$ could have been smaller for $k = 2$ or a higher $\alpha$ (i.e. a stricter distance condition), given $|V| = 750$. In this way, constants $\alpha$ and $k$ can offer control over $|V'|$, given any $|V|$.

Table 6 shows the mAP scores of the three queries that yielded the best and worst results. As expected, retrieval with SIFT features is very accurate for images with a lot of edges and corners (Figure 4.11), but various objects in the query image can interfere (Figure 4.12). This happened due to the way each query was processed; in our experiments, the user provides a whole image as query, rather than the object they are interested in.

<table>
<thead>
<tr>
<th>Top-3 Queries</th>
<th>Query</th>
<th>Average Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pompidou_4</td>
<td>0.851</td>
</tr>
<tr>
<td></td>
<td>pompidou_2</td>
<td>0.776</td>
</tr>
<tr>
<td></td>
<td>pompidou_5</td>
<td>0.761</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bottom-3 Queries</th>
<th>Query</th>
<th>Average Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>invalides_2</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>sacrecoeur_5</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>eiffel_3</td>
<td>0.084</td>
</tr>
</tbody>
</table>

Table 6: top and bottom 3 queries for $k=3$ and $|V|=200$
Figure 4.11: top-5 results of the top-3 queries for k=3 and |V|=200 (the first row depicts the query images)
The scores of the 55 queries are generally distributed more evenly for the $k = 3$ setting than $k = 1$ (Figure 4.13), with more queries scoring above 0.5 AP. For instance, for an initial vocabulary $V$, with $|V| = 200$, 15 queries scored above 0.5 AP with our method, whereas only 8 queries achieved this with the $k = 1$ setting. However, this observation is unlikely to be valid for larger initial vocabularies, unless constants $a$ and $k$ are properly adjusted.
Figure 4.13: histograms of AP score frequencies
4.3.2 Building the Inverted Index

Figure 4.14 shows the same inclination of our method towards a larger index as with the case of the Oxford buildings dataset. A similar observation to the one stated in 4.2.2 stands for the indexing time: for an initial vocabulary $V$ that contains up to 200 words, the time required to build the index grows linearly to $|V|$, and it is comparable to the standard method (see Figure 4.15).

| $|V|$ | $|V'|$ |
|---|---|
| 50  | 3,220 |
| 100 | 20,613 |
| 200 | 104,810 |
| 500 | 477,239 |
| 750 | 778,499 |

Table 7

**Index Size**

<table>
<thead>
<tr>
<th>size of initial vocabulary (centers)</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>750</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k=3$</td>
<td>68.5</td>
<td>82.3</td>
<td>103</td>
<td>129</td>
<td>143</td>
</tr>
<tr>
<td>$k=1$</td>
<td>42.7</td>
<td>44.5</td>
<td>49.6</td>
<td>54.2</td>
<td>55.7</td>
</tr>
</tbody>
</table>

**Indexing Time**

<table>
<thead>
<tr>
<th>size of initial vocabulary (centers)</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>750</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k=3$</td>
<td>0.67</td>
<td>1.12</td>
<td>2.52</td>
<td>7.79</td>
<td>15.5</td>
</tr>
<tr>
<td>$k=1$</td>
<td>0.58</td>
<td>0.9</td>
<td>1.74</td>
<td>4.06</td>
<td>6.34</td>
</tr>
</tbody>
</table>

Figure 4.14: index size for increasing initial vocabulary $|V|$

Figure 4.15: indexing time for increasing initial vocabulary $|V|$
Creating composite visual words from a small initial vocabulary is still more efficient than the standard method: contrary to $k = 1$ and $|V| = 750$, with $k = 3$ and $|V| = 200$, we can build a retrieval system with 3.75 times faster quantization and 2.5 times faster indexing:

$$Speedup_{QT} = \frac{QT_{|V|=750}}{QT_{|V|=200}} = \frac{nd3.75cl}{ndcl} = 3.75,$$

$$Speedup_{IT} = \frac{IT_{k=1,|V|=750}}{IT_{k=3,|V|=200}} = \frac{6.34}{2.52} \approx 2.51.$$

### 4.3.3 Query Processing

Each of the 55 queries was processed in the same way as in the previous dataset, and the results are similar. As long as $|V| \leq 200$, there is no significant difference between the two methods regarding the time required to retrieve the results of a query image (Figure 4.16). Again, these results are determined by the complexity of the implemented algorithm in Code 3, where a $O(n \log n)$ mergesort of the cluster centers is performed.

![Average Query Processing Time](image.png)

Figure 4.16: average time required to process each of the 55 queries
5 Conclusion

In this thesis, we have proposed a novel method for enriching the visual vocabulary of the bag-of-visual-words model. Composite visual words are generated from a small set of a few hundred initial words that were produced after the quantization step. Quantization was performed by clustering the features with k-means, resulting in an initial vocabulary where each word corresponds to the label of each cluster center. During indexing, each feature vector is assigned a composite word as the concatenation of the k initial words whose corresponding centers are closest to the vector in ascending order of their distance. Prior to the concatenation, each word must pass our proposed distance condition, which introduces a progressively tighter constraint as more words qualify, preserving the distinctiveness of the composite vocabulary. Finally, the collection of images is indexed as in text retrieval with a document of visual words for each image. Each query is issued as a whole image, from which composite words are produced in the same way and passed to the retrieval system as query terms.

Experiments on popular datasets revealed that our approach can accomplish decent mean average precision results. In particular, a setting of $k = 3$ nearest initial words from an initial vocabulary of 200 words attained 0.44 mAP, which was the highest result, outperforming the standard method where a single word is assigned to each feature vector. Moreover, such a small initial vocabulary introduces significant performance gains on quantization and indexing; a vocabulary of 500 words with the standard method reaches comparable retrieval quality to a composite vocabulary of the $k = 3$ nearest words from an initial vocabulary of as few as 50 words, which offers a Speedup $= 10$ on quantization time and 6.38 on indexing time. This way, a search engine can quantize and index a collection of images in significantly less time and still maintain adequate retrieval performance.

As future work, we consider experimenting on more and larger datasets, and producing more initial visual words. However, this implies more expensive quantization and indexing, exceeding the capabilities of a single desktop pc. Clusters of computers along with libraries for distributed computing, such as Hadoop [12], can offer the required processing power. We can also combine two datasets by building the visual vocabulary from features computed from the one and evaluating the performance on the other. In addition, a query can be processed by means of object retrieval rather than whole image retrieval, which enhances retrieval performance, since users specify the area in the image surrounding the object(s) of interest.

The process of assigning a feature vector to the nearest visual words is performed by an implementation of the algorithm shown in Code 3. This algorithm scanning through all centers, calculates the distance of each center and assigns the k nearest ones that meet the distance condition, after sorting them in $O(n \log n)$ time. Since it is obvious that this solution is not optimal, perhaps an approximate nearest neighbor approach would be more efficient. Especially for millions of extracted features, which is common even in small collections, significant time savings can be achieved. At the same time, it has been shown in [38] that the percentage of features assigned to incorrect nearest centers would be negligible. As
explained in 2.2.2, a randomized forest of k-d trees built over the cluster centers can be utilized to find the k approximate nearest centers to a feature vector.

An interesting point that arises from our work is the assignment of the appropriate values for the distance condition $\alpha$ and $k$, the number of centers that will be taken into account when forming a composite visual word. We believe that these values are highly correlated with the size of the initial vocabulary $V$: a large vocabulary requires a strict distance condition, (i.e. a high $\alpha$) and a small $k$, to avoid building an indistinctive composite vocabulary $V'$ of millions of words. On the other hand, it would be wise to have a small $\alpha$ and a high $k$ given a small initial vocabulary, in order to produce as many useful words as possible. As future work, an interesting problem would be to develop a method for automatically adjusting these two parameters, given the size of $V$.

In text retrieval, a thesaurus of synonyms can be utilized to return more relevant documents, as it correlates words having similar definitions. In image retrieval, the analogy of synonyms may be defined by the $k$ nearest neighbors to a visual word in the vocabulary, as long as it is ascertained that these words enhance the retrieval. An interesting problem would be to extend this analogy to cover composite visual words. To this end, the following approach is proposed to determine the visual synonyms for a visual word: given a composite visual word $w'$ comprising $k$ initial words $w_1, \ldots, w_k$, synonyms for $w'$ can be determined by the concatenation of two components: (a) the part of $w'$ until the $i$-th word, $i < k - 1$, and (b) a permutation of the remaining $k - i$ words. For example, if $i = 3$, the synonym for $w' = "ABCDE"$ is $s = "ABCED", \text{ where } A, B, \ldots, E$ are initial words. If $i = 2$, then $w'$ has five synonyms, equal to the number of permutations for the remaining $k - i = 3$ words, after ruling out the permutation that yields $w'$. In essence, parameter $i$ determines how abstract or specific the resulting synonyms shall be. The greater $i$ is, the less abstract the synonyms become. Of course, setting $i = 0$ does not return real synonyms.

Furthermore, stemming can be applied to composite visual words: given a word $w' = "ABCD E", \text{ words } "ABCD", "ABC", "AB" \text{ and } "A" \text{ can be considered synonyms because of the common stem } "A". \text{ Here, } i \text{ defines the length of the stem in words: if } i = 3, \text{ then words } "ABCD" \text{ and } "ABC " \text{ are synonyms of } w' \text{ because of the common stem } "ABC".$
References


