ImageTerrier: An extensible platform for scalable high-performance image retrieval

Jonathon S. Hare
jsh2@ecs.soton.ac.uk

Sina Samangooei
ss@ecs.soton.ac.uk

David P. Dupplaw
dpd@ecs.soton.ac.uk

Paul H. Lewis
phl@ecs.soton.ac.uk

Electronics and Computer Science, University of Southampton, United Kingdom

ABSTRACT

ImageTerrier is a novel easily extensible open-source, scalable, high-performance search engine platform for content-based image retrieval applications. The platform provides a comprehensive test-bed for experimenting with bag-of-visual-words image retrieval techniques. It incorporates a state-of-the-art implementation of the single-pass indexing technique for constructing inverted indexes and is capable of producing highly compressed index data structures. ImageTerrier is written as an extension to the open-source Terrier, "Terabyte Retriever", test-bed platform for textual information retrieval research. The ImageTerrier platform is demonstrated to successfully index and search a corpus of over 10 million images containing just under 10,000,000,000 quantised SIFT visual terms.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing

General Terms

Experimentation, Measurement, Performance, Algorithms

Keywords

Evaluation, Scalability, Image Indexing, Visual-terms

1. INTRODUCTION

Efficiently indexing large numbers of images or video frames for content-based retrieval purposes is a challenging task. In recent years, researchers looking at content-based image retrieval and automatic-annotation have begun to study techniques that allow image content to be treated in much the same way as we treat textual documents. For image retrieval, this allows the affordance of the use of optimised text-retrieval structures, such as the inverted index.

This paper presents ImageTerrier[1], an open-source software platform for researchers and developers to explore content-based image search. The ImageTerrier platform incorporates a highly-efficient state-of-the-art single-pass indexing engine, that is capable of building highly compressed indexes of image content. The retrieval side of ImageTerrier is configurable, and implements many recent techniques for incorporating the geometric consistency of matching visual-words into the scoring. A core feature of the ImageTerrier platform is that it is easy to extend to test new ideas and techniques. An early version of ImageTerrier won the first prize in the ACM MM open-source software competition 2011.

The aim of this paper is to highlight the novel aspects of the ImageTerrier platform and introduce other researchers to how the platform can be used to effectively test their ideas without having to re-implement a large body of underlying software. Whilst the separate technologies combined in ImageTerrier are well understood in their respective fields, ImageTerrier is novel because it brings together state-of-the-art techniques for scalable construction of highly compressed inverted indexes using the single-pass algorithm with state-of-the-art techniques for image retrieval using bag-of-visual-words models coupled with geometric consistency checking. In order to demonstrate the usefulness of the ImageTerrier platform demonstrate its scalability with a dataset of over 10 million images and also show how the platform can be extended.

The paper has the following structure: We begin by briefly surveying the history of the use of bags-of-visual-words and inverted indexes for image retrieval. Following this, Section 3 describes the state-of-the-art in textual search engine technology. Section 4 describes the ImageTerrier platform. Section 5 demonstrates the performance characteristics of the ImageTerrier platform through a number of experiments with corpus sizes in excess of 10 million images. Section 6 describes some example applications of ImageTerrier. Finally, the paper ends with some concluding remarks and discussion about future research on the ImageTerrier platform.

2. BACKGROUND AND RELATED WORK

The idea of using an inverted index structure for image retrieval tasks is at least 10 years old [see 14, for example]. The idea really took off when Sivic and Zisserman [13] pre-

[1] `http://www.imageterrier.org`

ICMR '12, June 5-8, Hong Kong, China

sented their seminal work which illustrated how descriptors of local features could be quantised into *visual words* and indexed efficiently using an inverted index structure. Fundamentally, Sivic and Zisserman's work broke the problem of image indexing and search into a number of analogies with the large body of work on text indexing.

Once an inverted index has been constructed, it can be searched very efficiently, and allows documents to be ranked against a query (see the next section for a discussion of how this works). Sivic and Zisserman used the cosine of the angle between the document and query as the measure of similarity, and also weighted both their visual words and documents using a scheme from the text-retrieval community called Term Frequency/Inverse Document Frequency (TF/IDF). Finally, they showed how precision could be increased by applying a re-ranking scheme to the top-matched documents using the spatial layout of the matching features in the query and retrieved documents. This spatial re-ranking also has a direct correlation with a technique in text search called *proximity search* in which the terms in the query are required to occur near each other in the retrieved documents. Over the years, a number of variations on Sivic and Zisserman's original approach have been described:

- **Different detectors and features**. Many different types of detector [e.g. 5, 7, 8] have been used. The most popular local descriptor is SIFT [5] due to its robustness.
- **Increased vocabulary size and hierarchical indexing**. In their original work, Sivic and Zisserman used vocabulary sizes of up to 12000 words learned with k-means clustering. More recently, researchers have used vocabularies of upwards of one million words learned using variants of k-means such as hierarchical [10] or approximate k-means [12]. In the case of hierarchical k-means, modifications to the inverted index through the inclusion of virtual postings lists are required to maintain precision [10] at the cost of a larger and slower index.
- **Different scoring and weighting schemes**. Experimental evidence has suggested that TF/IDF with cosine similarity is often sub-optimal for image retrieval. Alternatives such as the L1 distance with (optional) IDF weighting have been shown to work well [10].
- **Improved re-ranking schemes**. [13] originally used a loose spatial consistency approach that weighted matching words by the consistency of their neighbours. It is now common to see techniques that use strong constraints such as the fitting of affine transforms, planar homographies or even constraints based on the Epipolar geometry. Weaker constraints based on the scale and orientation of matching visual words have also been proposed [4].

In terms of implementation, all of the techniques described above have been tested using custom software, usually written solely for the task of performing the experiments required to verify a particular technique. Unfortunately, this is inadequate when we want to perform comparisons of techniques or explore scalability. As an example, Sivic and Zisserman used a Matlab sparse matrix (held in memory) to represent their inverted index; whilst this was fine for their experiments, it doesn't scale with corpus size or vocabulary size and is likely to be very inefficient. As a second example, to our knowledge none of the published work has actually addressed how the inverted index can be constructed for very large corpora, where the inverted index and visual-term features is bigger than the available memory. As we will see in Section 3, efficient construction of the index is vital for scalability. The ImageTerrier platform aims to provide a way for researchers to experiment with different variations of indexing and retrieval techniques at a range of scales, without having to re-implement large amounts of code.

There have been many open and closed source systems developed for image retrieval in the past. Systems such as Lucignolo and VIRaL allow search across pre-defined image corpora; not allowing custom corpora to be indexed. Solutions such as the GNU Image Finding Tool (GIFT)[2], the Flexible Image Retrieval Engine (FIRE)[3] and Lucene Image REtrieval (LIRe) [6][4] provide tools and frameworks with which custom image corpora can be both indexed and searched. ImageTerrier attempts to address many shortcomings of these existing systems. For example, although FIRE provides a large range of distance metrics it doesn't address the indexing problem and instead relies on linear scan which does not scale. GIFT does provide a custom implementation of a file-based inverted index and a two-pass indexing strategy, however the project is not currently maintained and is difficult to extend. LIRe has many similarities to ImageTerrier. LIRe builds on top of Apache Lucene (a text indexing library) in order to provide content-based image retrieval using a variety of in-built image features. For dense features LIRe relies on linear scan which has scalability problems, however it does support inverted indexing of BoVW using built-in SIFT and SURF features and k-means clustering. Other than adding additional image features, extending LIRe involves digging deep into the internals of Lucene. LIRe has no direct support for controlling index compression, scoring schemes and reranking schemes.

3. MODERN TEXT SEARCH ENGINES

As mentioned previously, we intend to use optimised data-structures from the text-retrieval field, including the inverted index, in order to build an efficient search and retrieval system. Briefly, a typical (text) retrieval engine consists of three major index components:

The Document Index. The document index is a record of the documents in a collection. Typically, each record in the index stores the numeric document identifier, the number of terms in the document and some additional metadata (such as the location or name of the document). The document index is typically loaded into the memory of the machine that hosts the search engine. It is common to keep the records ordered by their identifier, and numbered sequentially so that records can be looked up by direct addressing.

The Inverted Index. The inverted index is like an ideal book index. For every term in the corpus it stores the documents that contain the term and the number of occurrences in the respective documents (note that non-occurrences are

[2] `http://www.gnu.org/software/gift/`
[3] `http://code.google.com/p/fire-cbir/`
[4] `http://www.semanticmetadata.net/lire/`

not stored). Pairs of (documentId, frequency) are called postings. The postings for all documents containing a particular term is called a postings list. The inverted index is formed by appending the postings lists for all terms in the same order as the terms appear in the lexicon (see below). The inverted index is typically very large and is stored as a file that resides on a disk[5]. Postings lists are read directly from the inverted file on disk as required by a given query. Some inverted indexes are known as *augmented* indexes. Augmented indexes store additional information with each posting. A common use of an augmented index is to store the position of each term occurrence relative to the beginning of the document; this allows phrase and proximity searches to be performed.

The Lexicon. The lexicon is an index of the terms in the corpus. Each record of the lexicon contains the term, the frequency of the term in the entire corpus, and an offset that determines where to start reading the postings list for the respective term from in the inverted index (sometimes the end offset is used instead; sometimes both start and end are included). The lexicon, like the document index is also loaded into main memory for efficient access. The records are indexed by their terms through a hash-table or b-tree structure to ensure that records for a given term can be looked up very quickly. The lexicon might also contain a numeric identifier for each term.

3.1 Index Compression

The inverted index is usually heavily compressed to make it more efficient. On current computer systems the bottleneck for using the inverted index is the speed at which it can be accessed from disk. Reducing the amount of data that needs to be read through compression allows the index to be accessed faster. The cost of compression is that the CPU has to be used for decompression, however, modern processors are able to decompress the data many times faster than the rate at which the data can be read. Index compression is still also important if the inverted index is stored in memory as the amount of memory available is usually rather limited and compression allows a bigger index to be held. Fundamentally, the postings within an inverted index consist of a pair of integer numbers — the document identifier and the term frequency. The range of typical compression techniques is too large to list here, but a good introduction can be found in [15].

3.2 Single-pass indexing

The classical technique for constructing an inverted index uses two passes through the data. In the first-pass, a structure called a direct index is created. The direct index stores lists of (termId, termFrequency) for each document. In the second pass, for each and every term the direct index is scanned for occurrences of the term and the postings list is constructed and written to disk. If the document corpus is large, then the direct index (and inverted index) will not fit in memory, and will have to be written to disk. Unfortunately this means that the second pass will be very I/O intensive and thus rather slow. An alternative to two-pass indexing called the single-pass technique [3] solves some of the problems of two-pass inverted index construction. The single-pass technique works by processing documents and directly building the postings lists in memory. Once the available memory is exhausted, the postings lists are flushed to disk as a 'run'. The 'run' is essentially a sub-index over a portion of the documents in the corpus. Once the postings lists have been flushed and their memory has been released the indexer continues working through the corpus until the memory is exhausted again at which point another run is created. The process continues until all the documents have been processed. Finally, all the runs are merged on disk into the final inverted index. Single-pass indexing is considerably quicker than the two-pass technique [3].

[5]Google has recently moved to a distributed in-memory inverted index for its web search engine, however this requires massive amounts of hardware.

3.3 Querying

In order to perform a query and search the index, each of the query terms are looked up in the lexicon, allowing the postings lists to be retrieved and processed. Each posting in the posting list is considered in turn and a score is accumulated for each document identifier that is scanned. Many scoring techniques can be used which aim to favour some matches over others based on the definition of a "good match" given a specific task. A typical scoring method in text retrieval applies a weighting to the frequencies of each posting; the most common weighting scheme is called Term Frequency / Inverse Document Frequency (TF/IDF). After summation across all terms in the lexicon, such a weighted score results in an overall score proportional to the cosine distance. Once the processing is complete, the document identifiers can be ordered by decreasing score, and the document index used to look up the name/location of the actual document.

Sometimes, an additional pass is made over the inverted index in order to re-rank the best matching documents. For, example, in the case of *proximity search*, where the query is made up of a list of words that must appear near each other in the document, the second-pass over the index can be used to re-score the top documents using position information extracted from the relevant postings in an augmented inverted index. It is of course possible to combine the two search passes into one, but this isn't usually done in practice as the two pass technique works faster overall and requires significantly less memory for temporary data.

4. INTRODUCING IMAGETERRIER

The ImageTerrier platform implements the techniques for image retrieval described in Section 2 using the underlying ideas of efficient compressed index construction used by the text retrieval community. Rather than being written from scratch, ImageTerrier is built on top of an existing open-source textual search and retrieval system called Terrier[6]. Terrier is a high performance and scalable search engine that allows the rapid development of large-scale retrieval applications. The Terrier platform is designed to provide a comprehensive, flexible, robust, and transparent test-bed platform for research and experimentation in text retrieval [11]. In particular Terrier contains an efficient implementation of the single-pass indexing algorithm coupled with effective index compression.

The extension of Terrier to allow visual words to be indexed effectively was not straightforward. During the early

[6]`http://www.terrier.org`

parts of our investigation we benchmarked and analyzed many existing textual indexing solutions, including MG4J and Lucene. We chose Terrier specifically because it provides extensibility whilst still allowing tight control over index structure and compression. A specific example of Terrier's extensibility over the other solutions is the ease with which all the layers of software for dealing with text (i.e. tokenisation, stemming, etc) can be stripped away so we can deal with the abstract concept of a *term* which doesn't necessarily have to be textual in nature. In its raw form, however, Terrier lacks many of the key features required to implement state-of-the-art bag-of-visual-words based image retrieval. Firstly, we have made significant additions at the very base level of Terrier; In particular, we have added the concept of generic term payloads, meaning that visual words extracted from images can be indexed with their metadata (i.e. geometric/spatial information). In vanilla form, Terrier only supports storing a block number with each term (usually used to store the offset of the term from the beginning of the document). ImageTerrier allows any data to be stored and compressed (e.g. spatial location, scale, orientation, etc.). Without this, implementation of geometric consistency schemes using the spatial information of the respective visual term would be impossible. Secondly, we have used Terrier's extensible design to implement geometric consistency models and scoring techniques oriented towards image retrieval. These techniques are implemented as extensions of existing Terrier constructs (`DocumentScoreModifier` and `WeightingModel`). The use of Terrier as the underlying platform for ImageTerrier also allows additional affordances to ImageTerrier users. In particular, a number of features such as automatic query expansion, relevance feedback and document score modification based on external priors are all built in.

4.1 Project Structure

The ImageTerrier project is structured as both a library and a set of command-line tools. The ImageTerrier library does not itself provide feature-extraction and quantisation techniques, but instead allows features to be read from a flexible ASCII or binary format produced by other tools. These formats are supported natively by tools from the OpenIMAJ project[7]. Alternative formats can easily be supported through a custom subclass of the ImageTerrier `QLFDocument` class[8]. The assumptions made by the `QLFDocument` class are quite generic; it is assumed that the feature representing a document consists of a set of visual words, and each of those visual words may optionally have associated metadata (i.e. describing location information).

The ImageTerrier command-line tools allow a set of features saved as files to be indexed, and then allow the index to be searched. The indexing tool allows full control over the type of index that is created (for example, the type of location information embedded). The searching tool allows searches to be performed on an index with a given query document, and allows full control over the specifics of the search (including the scoring and weighting scheme [TF/IDF cosine, L1, L1IDF, etc] and any spatial-consistency technique).

In addition to directly indexing features, the indexing tool allows indexes to be built directly from images using standard features (i.e. difference-of-Gaussian SIFT, ASIFT, etc). When used in this mode, the tool can either load an externally produced vector quantiser, or create one on the features of the input images using standard techniques (k-means, approximate k-means, etc). To maximise speed, the indexing tool can take advantage of multiple processors. Indexes built directly from images have information on the type of feature and the data for the vector quantiser incorporated into themselves directly. This allows the searcher tool to be used directly with queries in the form of images and the extraction of the visual words to create the underlying query will happen automatically. The feature-extraction, clustering and quantisation features used in the tool are provided by libraries from OpenIMAJ. The ImageTerrier wiki has a walkthrough for using the command-line tools to build an index[9] and the tools are also briefly described in [2].

4.2 ImageTerrier Library Internals

Some details of the various indexing structures, scoring measures and re-ranking schemes is given below.

4.2.1 Compressed Augmented Inverted Indexes

ImageTerrier incorporates an inverted index structures that allows compressed payloads to be attached to each posting. This enables various types of of geometric consistency to be incorporated.

In the basic case, where no payload information is included, the postings list of each term is stored as pairs of document identifiers and frequencies. The postings in each list are ordered by increasing document identifier. Rather than storing the raw integer document identifier, the difference between the previous identifier and the current identifier is stored using gamma encoding. The frequency is stored using unary encoding.

ImageTerrier currently incorporates two kinds of payload that can be added to the postings. If the supplied payload implementations are insufficient, ImageTerrier is easily extended with new payloads and their encodings. Postings with payloads are written to the inverted index in the following format:

$$\Delta\text{documentId}, \text{TF}, [\text{Payload}_1, ..., \text{Payload}_{\text{TF}}].$$

Note that there is one payload written for every term occurrence and that by default, as with the no-payload index, the document identifier delta is written using gamma encoding and the term frequency with unary encoding.

The first payload structure implemented in ImageTerrier is the nearest-neighbour payload, which the information required to implement the geometric consistency scheme designed by Sivic and Zisserman [13]. The nearest neighbour payload stores a list of N spatially nearest neighbour words for each term occurrence. Each payload has the following format:

$$\text{numberNeighbours}, \text{termId}_1, ..., \text{termId}_{\text{numberNeighbours}}$$

Even though the number of neighbouring words included in the index (N) is set prior to index construction, the structure has to include the number of neighbours in each posting in case their aren't enough visual words in the image. The number of neighbours is written using unary encoding by default. The list of neighbouring term identifiers is sorted

[7] `http://www.openimaj.org`

[8] QLF stands for Quantised Local Feature

[9] `http://sourceforge.net/p/imageterrier/wiki/ImageTerrier%20Tools/`

in increasing order (with duplicates removed), and is written by using gamma encoding on the differences between consecutive term identifiers.

The second type of payload implemented in ImageTerrier is designed to store information on where (spatially within the image) each visual term occurred. This *position* payload stores a fixed length array of floating-point numbers for every term occurrence. The length of the array is set at indexing time. The array can be populated with a variety of things such as the x and y position of a visual term, the primary orientation, the scale, parameters defining an ellipse that describes the sampling region (i.e. for affine-invariant regions) or information about an affine-simulation. Rather than trying to write actual floating-point numbers to the index, the numbers are converted to unsigned integers by re-centering based on a preset minimum value and quantising based upon a preset maximum expected value, and preset number of bits for holding the integer in binary. This encoding allows the number of bits being used to store the payload to be kept at a much smaller level than if raw 32-bit floating-point numbers were used.

4.2.2 ImageTerrier Querying

In order to apply the geometric consistency technique using the augmented inverted index, a two-pass approach is suggested: Firstly, potentially matching images are retrieved using a variety of efficient weighting and similarity techniques. In the second pass the best of these retrieved images has its score modified to equal the total number of votes cast for all matching visual words between the image in question and the query. The number of images processed by the second pass is of course configurable.

Different scoring techniques are implemented as subclasses of the `WeightingModel` class. The `WeightingModel` is not responsible for calculating the complete similarity between a query and document, but is instead responsible for calculating the contribution to the overall score for a given term that has occurred in both the document and the query. Inside a `WeightingModel` you don't know what the actual term currently being processed is, but you do have access to all the relevant statistics about the term's occurrence in the query, the target and the entire corpus. This is sufficient for most weighting schemes. Terrier comes with a number of weighting schemes designed for text. ImageTerrier implements some extras ones including the unweighted L1 distance (the sum of the absolute difference between the number of occurrences of each term in the query and target), and the L1IDF measure [10] which extends the L1 distance with inverse-document-frequency weighting and provides some allowance for term significance in a given corpus.

As described previously, ImageTerrier also provides the ability to re-rank images after the initial scoring. Reranking schemes are implemented as classes which implement the `DocumentScoreModifier (DSM)` interface. ImageTerrier allows multiple `DSM`s to be run in sequence if desired, and of course it's possible to not use any score modifier at all. `DSM`s are given access to the index structure, the query and the current list of results, and are able to freely change the results in any way, although they usually only modify the score of each result item. It is easy to efficiently retrieve term-payload information for the documents in the result-set and the words in the query from within a `DSM`. In this way, implementation of geometric consistency techniques is possible. The currently implemented consistency techniques in ImageTerrier are shown in Table 1.

Table 1: Geometric consistency schemes implemented in ImageTerrier

Consistency Technique	Payload Type
Nearest Neighbours [13]	nearest-neighbour
Affine w/RANSAC	position with (x, y)
Homography w/RANSAC	position with (x, y)
Fundamental w/RANSAC	position with (x, y)
Consistent orientation [4]	position with orientation
Consistent scale [4]	position with scale

5. EXPERIMENTS

In this section we demonstrate the capabilities of ImageTerrier by generating the indexes required to efficiently search large datasets of images. We explore the time taken to generate and search these indexes as well as evaluating the performance of the indexes against a standard benchmark. Using the techniques outlined in this paper we can achieve state of the art retrieval results on datasets of over 10.9 million images with small retrieval times.

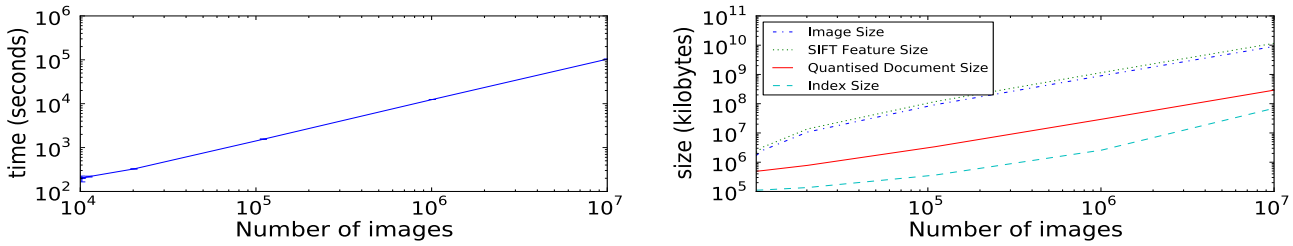
5.1 Experimental setup

The experiments performed to investigate the performance of ImageTerrier take the form of a traditional image retrieval or object recognition experiments. The UKBench dataset[10] [10] and evaluation protocol is used as the basis for the experiments presented here; the UKBench dataset consists of 10200 images of 2550 specific objects under varying orientation and illumination conditions. There are 4 images of each object in the dataset. The UKBench retrieval protocol is to take each image in turn as a query and calculate the four best matches (one, usually the first, of which should be the query image itself). A score is assigned based on how many of the top-four images are of the same object as the query. The score is averaged over all 10200 queries, and has a maximum value of 4.

The indexing technology we present here is capable of dealing with many more images than are present in the UKBench dataset. Therefore, to properly test the technologies we include a set of distractor images into the standard UKBench corpus. For this purpose we have downloaded all of the images found in the imageWordNet[11] [1] dataset. For our experiments, we include 0 to 10 million distractor images in increments of a power of 10 to show the effect of large image sets on indexing, searching and performance. In the experiments we perform the complete UKBench evaluation protocol 3 times for each combination of valid and distractor images. The indexes generated are the Basic type (i.e. without any extra geometric information). We present retrieval results using 3 score weighting schemes (L1, L1/IDF and TF/IDF) mentioned in Section 4.2.2. Starting with 0 distractors and adding more in powers of 10 up to 10 million distractors results in a total of 72 tests (8 combinations performed 3 times, each with 3 scoring strategies). Results and discussions are presented below.

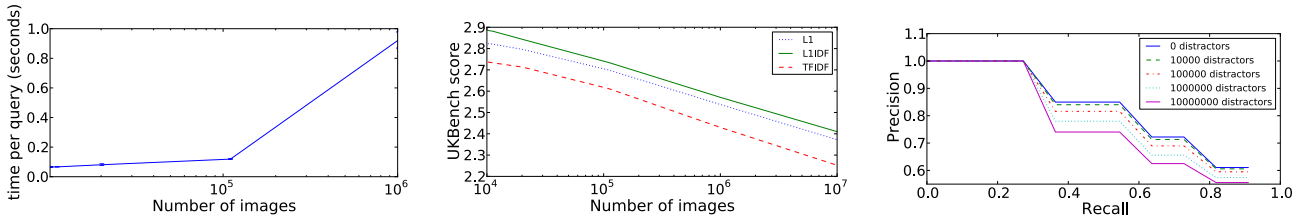
[10] `http://www.vis.uky.edu/~stewe/ukbench/`

[11] `http://www.image-net.org`



(a) (b)

Figure 1: (a) Time taken to index against total number of images being indexed; (b) Total size of index across number of documents as compared to original data sizes (images, sift features and quantised features)



(a) (b) (c)

Figure 2: UKBench Search Results. (a) Time taken for an average query against number of documents in the index. (b) Average UKBench score for 10,200 UKBench queries given number of distractors. Scores shown for 3 scoring schemes. (c) Average interpolated precision/recall curves for 10,200 UKBench queries against total number of documents. Scores shown for L1IDF scoring scheme.

5.2 Indexing

In Figure 1(a) we show index construction time as a function of dataset size and in Figure 1(b) we show the overall index size on the hard disk against original dataset size, quantised dataset size and index size per experiment. As the number of documents to be indexed increases, the amount of time taken to index increases. This relationships is linear due to disk IO remaining as the main bottleneck in the process. The overall size of the index on the disk has been shown to be roughly $\frac{1}{100}$th the size of the original dataset and $\frac{1}{10}$th the size of the quantised dataset. This difference is basically consistent regardless of dataset size though we can expect a lower limit due to the the lexicon size remaining constant (see Section 3). With a vocabulary of 1,000,000 words, the lexicon is roughly 3 times the size of the inverted index given 10,200 document and zero distractors. When 100,000 distractors are added, the inverted index becomes roughly 3 times the size of the lexicon. Through extrapolation we can estimate that with roughly 400 images this lower limit lexicon size overtakes the original dataset size[12].

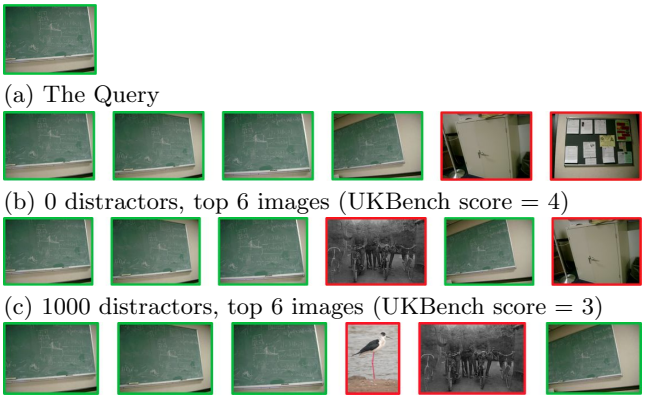
5.3 Searching

In Figure 2(a) we show the average time taken per query per experiment. The first thing that should be noted is that, once the index is loaded and initialised, a query is likely to take under 1 second with indexes of up to 1 million documents. This is comparable to the query times reported by Nistér and Stewénius, but it should be noted that our inverted index is completely disk-based, whereas Nistér and Stewénius's was held in RAM. The results show a near-linear increase in the time per query as a function of the number of documents (note the x-axis is logarithmic). This is to be expected as even in an inverted index, the postings list of a given term is likely to increase linearly with number of documents and in turn so will the time taken to score documents. However, the key improvement promised by the inverted index approach comes not in a reduction of the complexity of the algorithm, but rather the gradient of the linear complexity. The inverted index strategy promises a line of much lower gradient than a brute force strategy. We also expect a further drop in this gradient when the searching strategy is extended to work in a distributed manner.

The graph does not show the results for the index containing 10 million images. The average query time for this index increased dramatically to roughly 28 seconds per query. Further investigation showed that due to relatively large queries and no efforts made towards the use of stop words many images were scored for any given query. This results in IO saturation and heavy memory usage. Several approaches can be investigated to improve the performance for larger indexes. Using the term statistics to identify a set of stop words could result in fewer postings being interrogated and fewer documents scored per query. Furthermore a distributed search strategy through the use of a *sharded* index could allow the search of larger corpuses in smaller amounts of time.

In Figure 2(b) we show the UKBench evaluation performance as distractors are added and Figure 2(c) shows the average interpolated precision recall curve across all queries. We notice that the scores fall as distractors are added. However, this is to be expected as within UKBench, images which only tentatively achieved high scores with images that did not match very strongly can be easily confused when more images are added. An example of such a situation is

[12]Assuming 400 640x480 colour jpeg images each between 100k and 200k in size



(a) The Query

(b) 0 distractors, top 6 images (UKBench score = 4)

(c) 1000 distractors, top 6 images (UKBench score = 3)

(d) 100000 distractors, top 6 images (UKBench score = 3)

Figure 3: Example query and top-ranked results with increasing distractors.

shown in Figure 3 which shows the same query with 0 distractors, 1,000 distractors and 100,000 distractors. It should also be noted that the best score we demonstrate without distractors is 2.9 which is slightly below the state of the art. However, for these experiments we elected to use a more general vocabulary trained against the MIRFlickr25000 dataset, unrelated to both UKBench and ImageNet. Using UKBench itself to create a vocabulary from SIFT features we achieve scores near 3.2 with a 1,000,000 term vocabulary (compared to 3.16 reported by [10] using a non-hierarchical vocabulary from MSER-SIFT features). We achieve our best results with a score of over 3.65 by using ASIFT features [9].

6. EXAMPLE APPLICATIONS

The ImageTerrier technology provides an open source toolkit for the quick indexing and retrieval of images from large-scale collections with state of the art precision. This technology has been successfully applied in two demonstration applications giving a flavour of the kinds of problems which this toolkit can address. In this section we outline those applications and the role ImageTerrier has played within them.

6.1 GeoLocation

A recent trend amongst image sharing communities is to "geotag" images. That is, ascribe the location of the photographer to the photo taken thus giving some idea of where in the world a given photographic scene takes place. The geographical latitude and longitude may be ascribed by the photographic device itself which may be fitted with a GPS device. However due to power considerations and other limitations these annotations may be relatively inaccurate. These tags may also be ascribed manually by the user as a post processing step, in which case they may be quite inaccurate, limited to a rough pin placed manually on a map or even a place name annotation. There is a growing need for the ability to automatically annotate the geographic location of an image based on its visual signatures.

The process of automatically tagging a query image with geographic location works by firstly identifying a set of previously tagged images as being similar to the query image. As each individual image's geographic location may contain some inaccuracy it is important that large set of visually accurate results are returned whose geolocations can be geographically clustered in order to most accurately estimate the possible location of the query.

ImageTerrier provides the ideal platform upon which to build a large scale geolocation application, a demo of which can be found online[13]. Using flickr, a set of 150,000 images with geotags were downloaded for a specific region in the world, in the demo case for the Trentino region in italy. SIFT features were extracted from each of these images and the features were quantised using a vocabulary containing 1 million words. Using ImageTerrier, an index was constructed which also contained scale and orientation position payloads (see Section 4.2.1). Using this index, visually similar geotagged images to the query can be quickly returned. The images are then clustered using their geolocation, and a centroid of the largest cluster is used as the estimate for the geolocation of the query. Including the extraction, quantisation and search, a query image can be geolocated in under 3 seconds.

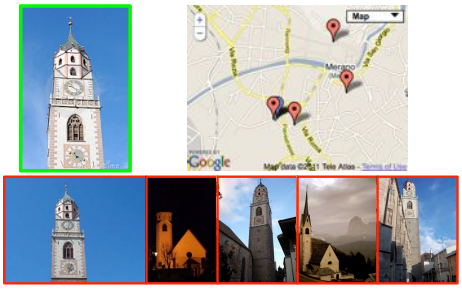


Figure 4: Untagged query image of a church successfully geolocated using ImageTerrier.

6.2 Stock Image Finder

Stock images are a common resource used by many types of media, both on the internet and in print, to add context to an article or news piece. Though sometimes cited, it is often difficult to find the original source of a given stock photo. Through identification of the original source it is possible for users to understand the context of the photo they are being presented with and therefore make a more informed decision about the information they are receiving. For example, a news vendor may choose to illustrate a news piece on a military conflict using imagery from a completely unrelated conflict. By doing so they may unfairly corroborate their article with spurious evidence.

This sort of behaviour is often difficult to discover manually. Using ImageTerrier it is possible to facilitate users with automatic methods of stock image source discovery. Over a period of 3 days a small dataset of 25,000 thumbnail images was collected from the Getty RSS feeds. The SIFT features were extracted and quantised using a vocabulary of 1 million visual words and indexed using ImageTerrier. A web service was created which was usable with a javascript bookmarklet[14] which allowed all the images on any given webpage to be searched against this index. Though 25,000 images makes for a limited set of potential articles which can be searched in this way, through consistent updating

[13] `http://geoimages.imageterrier.org`

[14] `http://stockphotofinder.imageterrier.org`

and merging of new indexes it would be possible to make a service usable on new Getty images as they are posted.

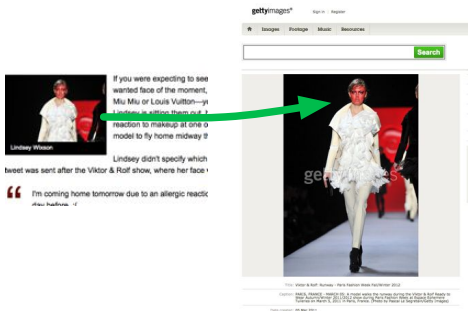


Figure 5: Original source of an image in a web article successfully located using ImageTerrier

7. CONCLUSIONS AND FUTURE WORK

The ImageTerrier platform provides a novel high performance feature indexing strategy underpinning a range of state of the art tools allowing researchers and developers to experiment with modern image retrieval techniques. In particular, it allows researchers to easily experiment with new techniques for scoring and implementing geometric consistency modifiers, and affords its users full control over the information stored in the inverted index. The platform also incorporates all of the advantages of the underlying Terrier platform, which includes numerous techniques that can be applied to image retrieval, such as automatic query expansion and relevance feedback. To our knowledge, there exists no other publicly available platform for BoVW-based image retrieval that supports scalable and efficient single-pass indexing to create highly compressed indexes whilst allowing a high level of control over the search process.

At the time of writing, we're currently experimenting with a scalable distributed indexing scheme based on Apache Hadoop. Whilst the underlying Terrier platform includes support for Hadoop indexing, it cannot be used with ImageTerrier indexes that support payloads. In the spirit of open-source development, features that are implemented in ImageTerrier that also have utility in text search are being pushed back into the underlying Terrier project; some parts of ImageTerrier that go towards supporting arbitrary term payloads have already made there way in to the latest Terrier release, we hope that we will shortly be able to push back the remainder of the changes for this, together with the new Hadoop indexing implementation.

8. ACKNOWLEDGMENTS

The described work was funded by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreements n° 270239 (ARCOMEM), 231126 (LivingKnowledge) and 287863 (TrendMiner) together with the LiveMemories project, graciously funded by the Autonomous Province of Trento (Italy).

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[2] J. S. Hare, S. Samangooei, and D. P. Dupplaw. OpenIMAJ and ImageTerrier: Java libraries and tools for scalable multimedia analysis and indexing of images. In *Proceedings of ACM Multimedia 2011*, MM '11, pages 691–694. ACM, 2011. ISBN 978-1-4503-0616-4.

[3] S. Heinz and J. Zobel. Efficient single-pass index construction for text databases. *J. Am. Soc. Inf. Sci. Technol.*, 54:713–729, June 2003. ISSN 1532-2882.

[4] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *Proceedings of ECCV 2008*, ECCV '08, pages 304–317, Berlin, Heidelberg, 2008. Springer-Verlag.

[5] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, January 2004.

[6] M. Lux and S. A. Chatzichristofis. Lire: lucene image retrieval: an extensible java cbir library. In *Proceedings of ACM Multimedia 2008*, MM '08, pages 1085–1088. ACM, 2008. ISBN 978-1-60558-303-7.

[7] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In P. L. Rosin and A. D. Marshall, editors, *BMVC*. British Machine Vision Association, 2002. ISBN 1-901725-19-7.

[8] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool. A comparison of affine region detectors. *IJCV*, 65(1/2):43–72, 2005.

[9] J.-M. Morel and G. Yu. ASIFT: A New Framework for Fully Affine Invariant Image Comparison. *SIAM J. Img. Sci.*, 2009.

[10] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *CVPR*, pages 2161–2168, 2006.

[11] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A High Performance and Scalable Information Retrieval Platform. In *Proc SIGIR*, 2006.

[12] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, 2007.

[13] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *ICCV*, pages 1470–1477, October 2003.

[14] D. M. Squire, W. Müller, H. Müller, and J. Raki. Content-based query of image databases, inspirations from text retrieval: Inverted files, frequency-based weights and relevance feedback. In *PATTERN RECOGNITION LETTERS*, pages 143–149, 1999.

[15] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes : Compressing and Indexing Documents and Images*. Morgan Kaufmann, San Francisco, CA, 2. edition, 1999.