A SYSTEM FOR COLOR BASED IMAGE RETRIEVAL

TOM GIBARA

Note 0.1. This is a very preliminary version of a document that I am currently writing. My intention is that the next revision will provide a much more coherent presentation of the ideas behind the system and contain many more details. I am publishing it in its present state to coincide with the public availability of its implementation;[1] someone might find something interesting in this document even in its current reduced state.

1. Introduction

Publishers, graphic artists and web designers are among those who often need to find imagery matched by both colour and theme to a particular brief. Stock photography websites provide photographs and other imagery for use by such professionals. Many have provided thematic searching using keywords (or tags in the Web 2.0 parlance) for some years and some sites have now added colour based searching where a single colour may be specified together with other parameters.

In providing a more flexible color based image search than single colour searching, there are a number of technical considerations that are not present with more commonplace searches. A fully fledged image search needs to provide users with access to an extremely large number of images[2] and with controls that allow them to search for multiple colours at a variable degree of accuracy and abundance. Text based online searches over large numbers of documents typically use an inverted index[3] but these requirements are not well suited to such an implementation.

This document describes the preliminary implementation of an image search system that seeks to address these issues. The system has the following characteristics:

- Search time is proportional to the number of images in the database, but the constant of proportionality is extremely low.
- The storage required to index image colour data can be tailored to requirements and typically will not exceed 128 bytes per image.
- All elements of the system are easily parallelized.
- Multiple colours may be searched for in a single scan of the index.
- There is no significant memory usage during any phase of the system.

The system operates in three distinct phases. During the extraction phase, colour data is drawn from a collection of images after which indexing occurs at which point the extracted information is condensed for rapid retrieval. After these two phases, searching is performed by scanning the index to identify images that contain specified colours. Each of these phases is described separately in the sections that follow after which more information is given about the current implementation and its future potential.

2. Extraction

Color information is extracted from a sequence of images by processing each image in turn. All images are processed in an unspecified RGB colorspace. The pixels of each image are clustered using a fast spatial clustering algorithm[4] on their colorspace coordinates. The clustering algorithm employed is potentially sensitive to bias in the order in which elements are presented for clustering. The typical scanline coherence exhibited by images may provide just such a bias. To overcome this, pixels in the image are ordered by a fast de-coherence algorithm[5] prior to clustering.

The clustering algorithm has negligible memory requirements, scales linearly with the number of pixels in the image but requires a maximum number of clusters to be specified, which will effect performance. For this reason a modest maximum number of colours is specified, though this is not problematic since for this application, only the dominant colours are required.

As a natural consequence of its execution the algorithm generates data on the location, variance, abundance and mean of the colour clusters it identifies, though at present only the last two of these are used in the system.

After extraction this colour data may either be indexed immediately or stored for subsequent indexing.

Date: 5th September 2007.

[1] `http://www.tomgibara.com/computer-vision/color-search/`

[2] Stock photography companies frequently carry 2 million or more images.

[3] `http://en.wikipedia.org/wiki/Inverted_index`

[4] `http://www.tomgibara.com/clustering/fast-spatial`

[5] `http://www.tomgibara.com/computer-vision/minimizing-spatial-cohesion`

3. Indexing

The accumulated color information is compiled into an index for fast searching. Each image is assumed to have a unique identifier with a bounded length binary representation. This identifier is stored in the index together with the color information which is encoded using a fixed-length Bloom filter.[6] Each index record is thus the pairing of an identifier and a filter; both fields are fixed-length, therefore the record length is constant. This facilitates fast linear scanning of an index that can be stored in-memory or on-disk as a simple fixed-length record array.

The colour information for a given image consists of a small colour map that records the abundance of each extracted colour. This map of colours is pruned by removing the least abundant colours. The remaining elements are classified into a small number of bands, based on their abundance. A set is then created to which each colour is added at varying fixed degrees of quantization; this allows for approximate colour matching.

The resulting set is converted into a fixed-length byte array (irrespective of its size as per the characteristics of Bloom filters) and appended to the index, followed by the image identifier. In principle, new records can be added to any existing index in this way and removed by scanning the index for an image identifier and zeroing the bits of its color data.

4. Searching

A search on an index begins with a set of colours that are being sought within an image. A minimum quantization and minimum abundance may have been specified for any or all of the colours specified; in the absence of specified minimums, the least possible value is assumed. For each colour, the abundancy bands and quantization degrees that equal or exceed its specified minimums are enumerated and individual Bloom filters are constructed for each combination.

The result is a set of Bloom filters which match each colour with a desired abundance and accuracy (degree of quantization). Each of these filters is then bitwise compared with each filter in the index. The result of each comparison can be scored individually with more accurate (less quantized) and more substantial (more abundant) matches scoring more highly. These individual scores can then be combined to create an overall score. In principle, the scoring policy can be chosen individually for each search. Records which contain every colour in the query and which exceed a minimum score result in a matching image; the score may be used to order search results for presentation of the best matches first. The associated image identifier may be used to retrieve further information about the image for presentation to the user.

[6] `http://en.wikipedia.org/wiki/Bloom_filter`

5. Current Implementation

At present, the system is implemented as a set of three small Java applications that each execute one phase. The applications are all run using the Sun Java 6 Runtime under Windows NT Professional on a 2.1Ghz Intel Core Duo workstation.

The first application uses the Flickr API to obtain a list of interesting[7] photographs. One medium-sized[8] image is downloaded for each photograph from which a maximum of 15 colours are extracted using clustering as described above; the colours are obtained from the cluster centres, and their frequencies from the cluster sizes. A unique 64 bit identifier (Java long) is assigned to the image and, together with the colour data and some Flickr URLs, is stored in a PostgreSQL database. This process typically takes between 0.5s and 1s per photograph.

A second application uses JDBC to retrieve all the images and their associated colours from the PostgreSQL database, ordered by their identifier. Images are classified into 5 abundancy bands (3%-20%, 21%-40%, 41%-60%, 61%-80%, 81%-100%) and quantized at three levels by removing the last 2, 4, and 6 bits of their 8 bit RGB colour coordinates. Each set of colours is encoded into a 128 byte Bloom filter and, together with the image identifier, written sequentially to an index file. The 1024 bit filter uses 11 independent hashes drawn from a SHA512 digest. To date, approximately 44,000 images have been recorded in the database and subsequently indexed. Performing the SELECT statement takes 7.5s and indexing the returned rows a further 25s.

The final application embeds Jetty to provide an extremely simple web based search system. A set of colours is parsed from the HTTP query string, a search constructed from those colours, and HTML page generated from the search results. Searching is conducted by scanning the color index file as outlined in the previous section. Scoring uses the formula: $(a+1)2^b$ where $0 \leq a \leq 2$ is the ordinal of the accuracy level and $0 \leq b \leq 4$ is the ordinal of the abundancy band. Scanning the index and ordering the search results typically takes from 60ms (for a single colour search) to 170ms (for a four colour search).

As yet, no optimization has been performed on any of the above applications. In particular, the search application performs no caching - all data is read from disk on each request. No attempt has been made to introduce parallelism into any of the processes.

6. Future Work

In no particular order, the following additional work could be undertaken to improve the search system:

[7] Interesting here is defined by the Flickr website

[8] In this context medium-sized means that neither dimension exceeds 500 pixels

- Improving the search interface; allowing users to weight searches for accuracy/frequency.
- Adding colour localization; the clustering algorithm can be easily extended to provide a physical centre for the colours it clusters, this could be recorded in the index and used in searches.
- Combining colour extraction with other image analysis algorithms; composition, shapes, textures and other image features could be extracted.
- Unifying with searches on photo tags; the Bloom filters operate independently of the set elements they encode, Flickr tags could easily be added to the filters and the search could combine tag and colour searching in one index.
- Taking advantage of parallelism; the index can easily be split into multiple segments that are independently queried.
- Implementing index file caching; for small indices such as the current one, it should be an option for the index to be stored in memory.
- Analyzing performance metrics; various system components need to be analyzed for their performance characteristics, especially at scales comparable to real usage scenarios.
- Identifying best bands and levels; the granularity of color-frequency and color-quantization that users require needs to be determined through experiment.
- Mutating the index; it should be possible to adjust the index (by adding, removing or modifying records) without rebuilding the entire index.
- Partitioning the index; by using bit-pattern techniques (or others) the index can be decomposed into (possibly overlapping) sub-indices that can be individually scanned - this is vital to ensure sub-linear search times.