CONTENT-BASED IMAGE RETRIEVAL SYSTEM

For Michigan State University CSE484

Zach Riggle
riggleza@msu.edu

Chris Van Wiemeersch
vanwieme@msu.edu

TABLE OF CONTENTS

CONTENT BASED IMAGE RETRIEVAL

With FLANN and Lemur

Zach Riggle
Chris Van Wiemeersch
Fall 2010

Approach

Our approach to building *CBIR (pronounced* ***sea-bird****)*[1] followed the instructions provided to the class with the following differences:

- We clustered to convergence (-1) instead of 15 clusters
- We used the latest version of *FLANN* (1.6.5) instead of the provided version
- Our original *"Mark I"* was implemented in pure Python, for ease of implementation and extensibility.

Test Query Images

Approach In-Depth

The *esp.feature* file is processed into a large array that is passed to *FLANN* for clustering, per the instructions. These clusters are saved to file via the provided *FLANN* functionality, which saves the clusters to the HDF5[2] format.

[1] While CBIR is the name for the mechanism by which the images are retrieved, we also selected it as the *name* of our project by stylizing the pronunciation.

[2] HDF5 is a unique technology suite that makes possible the management of extremely large and complex data collections.

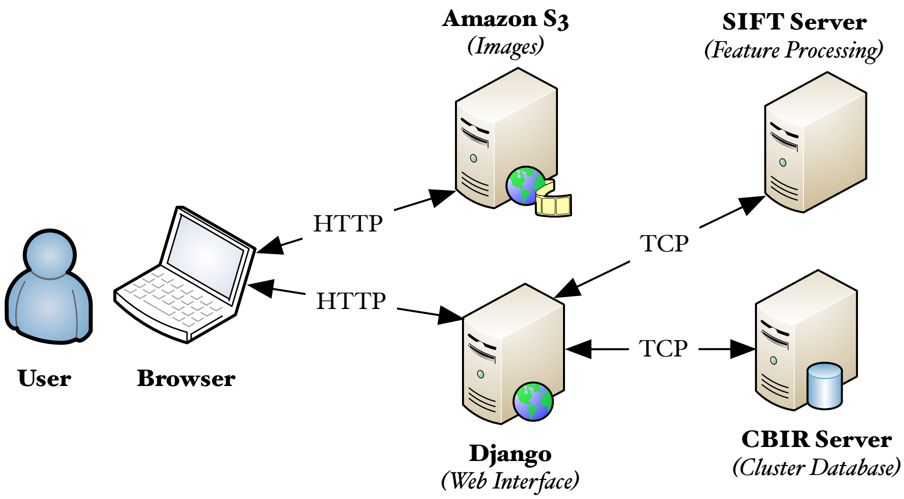
We then perform a nearest-neighbors search against the clusters, with the features as input. This provides us with the "*bag of words*" representation. This information is saved to a series of files outlined in the project description.

Lemur is used to index the data, using the instructions provided for the previous course exercise. The *RetEval* command is used to evaluate each query. The results of the query are then displayed to the user.

Components

Several different components interact to allow our query system to operate together. Each piece plays its part in the overall "big picture".

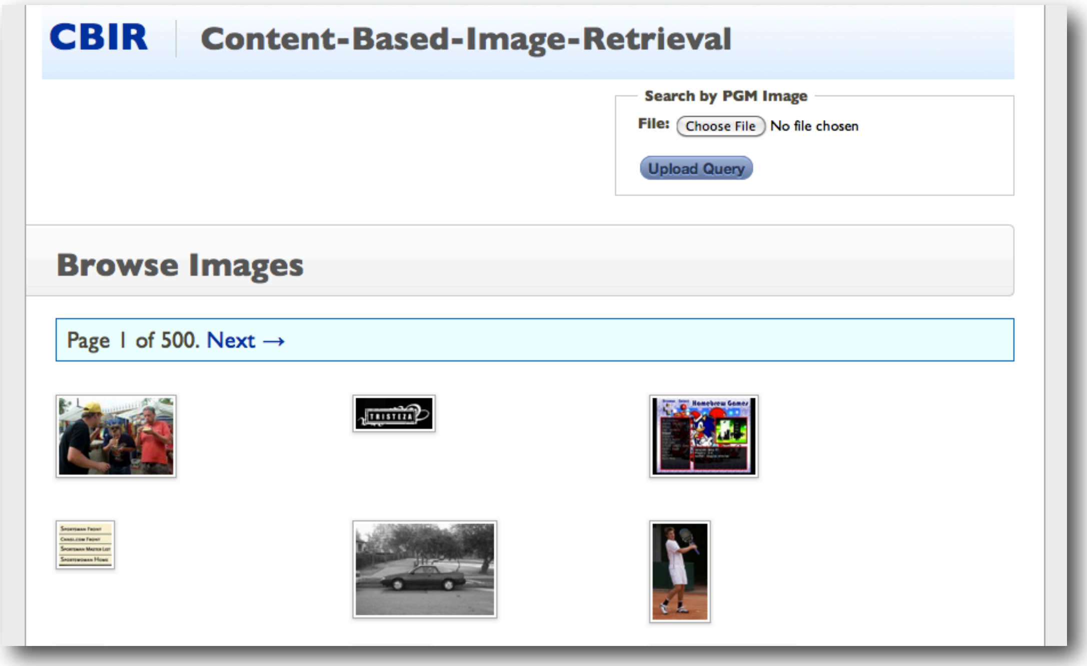
- Django GUI (Python)
- CBIR server (C++)
- SIFT server (Python)



CBIR System Diagram

Django GUI

The *Django GUI* ties together the *SIFT* and *CBIR Servers* with a web interface, allowing users to upload files for querying.



The Django GUI

Using common routines developed for querying the *CBIR* and *SIFT Servers* from the command-line, the *Django GUI* displays the results in a much more graphically pleasing manner.

The *Django GUI* is also responsible for shuttling data back and forth between the *CBIR* and *SIFT* servers, as well as invoking *Lemur* and processing its output.

CBIR and CBIR Server

CBIR is the name of the C++ executable used for loading the *esp.feature* file, processing it into memory, performing the clustering, cluster-indexing, and bag-of-words generation.

The *CBIR Server* uses the same C++ code-base as the clustering and bag-of-words generation code. In fact, it is the same binary, but performing a different functionality. The clusters are loaded into memory, and indexed. The program then listens for connections on a specific port, and receives data in the same format as the course-supplied *esp.feature* file. The program then performs a nearest-neighbor search against the provided features, and returns the cluster numbers from the search.

After the nearest-neighbor search has completed, the *CBIR Server* returns that data formatted as a *Lemur* query. This query is then processed by Lemur's *RetEval* command.

SIFT Server

Since our development platform was Mac OS X, we were not able to run the *SIFT* program directly on our machines. In order to rectify this, we constructed a small Python wrapper that receives a PGM file, prcesses it with *SIFT*, and performs a minor amount of processing on the output data to get it into the same format as *esp.feature*.

This information is then passed onto the *CBIR Server* (above), where it is converted into a Lemur query according to its bag-of-words representation.

Innovations

A number of non-standard features made their way into our project.

Command-Line Querying

In order to facilitate development without a GUI, a command-line querying tool was developed, *query.py*. The name of the target file(s) are provided on the command-line, and the resulting images are opened in the user's web browser. The images are all hosted on *Amazon Simple Storage Service (S3)* so that they are accessible even to users who do not have the library of actual images installed.

Pre-Query Indexing

The *CBIR Server* indexes all of the clusters before accepting any queries, so that when performing the nearest-neighbor lookup, the operation is quick and responsive. Ultimately, the user experience is greatly enhanced.

Cloud Computing

Our platform is currently running on the *Amazon Elastic Compute Cloud (EC2)* and the images are served by the *Amazon Simple Storage Service (S3)*. By leveraging cloud computing, we can expand our capabilities and capacity simply by allocating new *EC2* instances. Content distribution is handled by *S3* so that images load quickly without regard to the user's physical location.

In addition to simply running on *Amazon EC2*, the individual components of our CBIR system were designed to be interchangeable and distributable. For example, multiple nodes could run the *CBIR Server* (since this requires the bulk of the processing power) while only one web server instance could be required. The *Django GUI* could perform a round-robin rotation between these multiple *CBIR Server* nodes, resulting in shorter query times under load.

Enhanced Clustering

Instead of settling for the standard number of clustering iterations, we allowed *FLANN* to continue clustering until convergence was reached.

C++ and Python Implementation

Our original implementation was in pure Python, utilizing *pyflann*. Ultimately, this approach was scrapped as the Python-FLANN translation layer made the clustering operation too slow to be acceptable.

The feature-processing, cluster-building, cluster-indexing, and bag-of-words generation system was re-written in C++. We leveraged several different technologies to make this quick and easy, such as the *Boost* libraries.

OPEN SOURCE

Our entire project is released as an open-source project[3] [4], so that future students can learn from our successes and mistakes, as well as have a reference implementation to compare theirs to.

The project is fully documented, and includes instructions on how to set up prerequisites, build, install, and use the complete query system.

[3] CBIR, CBIR Server and SIFT Server: http://goo.gl/VIJzp

[4] Django GUI: http://goo.gl/IGrBh