View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Scholarworks@GVSU

Grand Valley State University

ScholarWorks@GVSU

Technical Library

School of Computing and Information Systems

2015

An Evaluation of Oracle's StillImage Plugin as a Platform for Dynamic Image Search

Nicholas Nelson

Grand Valley State University

Follow this and additional works at: https://scholarworks.gvsu.edu/cistechlib

ScholarWorks Citation

Nelson, Nicholas, "An Evaluation of Oracle's StillImage Plugin as a Platform for Dynamic Image Search" (2015). *Technical Library*. 218.

https://scholarworks.gvsu.edu/cistechlib/218

This Project is brought to you for free and open access by the School of Computing and Information Systems at ScholarWorks@GVSU. It has been accepted for inclusion in Technical Library by an authorized administrator of ScholarWorks@GVSU. For more information, please contact scholarworks@gvsu.edu.

An Evaluation of Oracle's StillImage Plugin as a Platform for Dynamic Image Search

By
Nicholas Nelson
August, 2015

An Evaluation of Oracle's StillImage Plugin as a Platform for Dynamic Image Search

By
Nicholas Nelson

A project submitted in partial fulfillment of the requirements for the degree of

Master of Science in

Computer Information Systems

at

Grand Valley State University

August, 2015

D. Robert Adams __12/19/2015__ **Date**

Table of Contents

Abstract

In this project, we look at some different off-the-shelf solutions for creating an image search engine that, given an unknown cover for a comic book, would return relevant information about a matching comic book in the database. The project first examines some alternative tools available to developers to do image search and comparison, and then looks at Oracle's solution for image comparison, Still Image. A prototype search is built to test Oracle's capabilities and is evaluated on speed and quality of results. In the end, Oracle is determined to be a useful tool for storing and comparing images, but is still not capable of searching for an image in a reasonable amount of time.

Introduction

Many comic book fans can relate to the experience of standing in a comic book shop with a comic book in hand and needing to find out some information about the comic. With modern smart phones, comic book fans have been able to use the device in their pocket to do searches on the Internet to find the information about a comic book. There is a wealth of information that a fan might be looking for while holding that comic book in his/her hand. First, there is the obvious question of "do I already have this issue in my collection?" Many fans have large collections and it is impossible to keep a mental catalogue of every issue. Second, a fan might have been drawn in by the characters or the name of the comic, but might not know anything about the issue in question. Where does this issue fit into the series? What year was the book published? Does this issue have positive or negative reviews? Is this issue worth any money? Finally, a fan might have decided they want to buy this issue and want to quickly be able to add it to an application that keeps a digital catalog of all of the issues they own.

All of these scenarios can be solved using a smart phone and apps as they exist today. However, in all cases, the user is responsible for doing the majority of typing of information in order to search on the Internet. This project proposes a simple way for a user to take a picture of a comic book cover and instantly have all of the information pulled up for that issue, with the ability to add it to a digital catalog.

More specifically, we explore whether or not off-the-shelf components can be used to create an image search application within the limited domain of comic book covers. In order to accomplish this goal, we will examine several different technologies that can be used to create an image search application and determine if they are viable options for our comic book cover image search engine.

Background and Related Work

There are several different existing products and technologies that can be used for doing searching and comparisons of images.

An obvious first choice is Google's Custom Search API [Google Developers 2015]. Google is effective both at finding images that match a certain term and also at matching images that look similar to an unknown image. In our testing, Google was often able to find the correct issue on existing comic book database site ComicVine.com when given a random comic book cover. However, since a certain cover could appear on several different pages in a website, it is hard to predict what content will be available at a matching site returned from the search engine. Additionally, Google's Custom Search API does not allow a developer to create an app that sends an unknown image to the API. The API only works when providing text queries. The final drawback to the Google API is the cost. The first 100 queries per day are free, however, after that, it costs $5 per 1000 with a max of 10,000 queries per day.

Another option was a 3rd party image search called TinEye.com [Tineye.com 2015]. However, there were several downsides to this solution as well. First, in testing, the site seems to try to match a photo to an object more than it does find similar images. That is, given a picture of a pencil, it will return that the picture contains a pencil. However, it will not find other pictures of a pencil. In addition to this major downside, the site actually does not have a non-commercial API, meaning it would also be expensive to implement.

Another alternative search engine for this project was the Wolfram image search engine ImageIdentify.com [Imageidentify.com 2015]. However, in testing, it was never able to even come close to matching any comic covers given to it. This is still a relatively new product from Wolfram, so it could potentially get better with time.

The final option would have been to use a open source image comparison library, such as the Resemble.js library [Cryer 2015]. This library is very effective at comparing 2 images. However, in order to create a database of images, we would have to use this software to run a comparison against every image in our database. This would be a time consuming process and would not be fast enough for a commercial application.

Solution Design

Given our budget and time limitations the comic book database prototype was designed using the Oracle DBMS with the Still Image [Docs.oracle.com 2015] plug-in installed. The Oracle instance that we used for development and testing of the prototype was an Oracle 12c virtual machine [Oracle.com 2015] running on the VirtualBox platform on an Apple MacBook Pro. The plug-in seemed to have the necessary features to perform image search. The design is straightforward: load the database with known images, then determine if the Still Image plug-in can find similar images given a query image. This is shown in Figure 1.

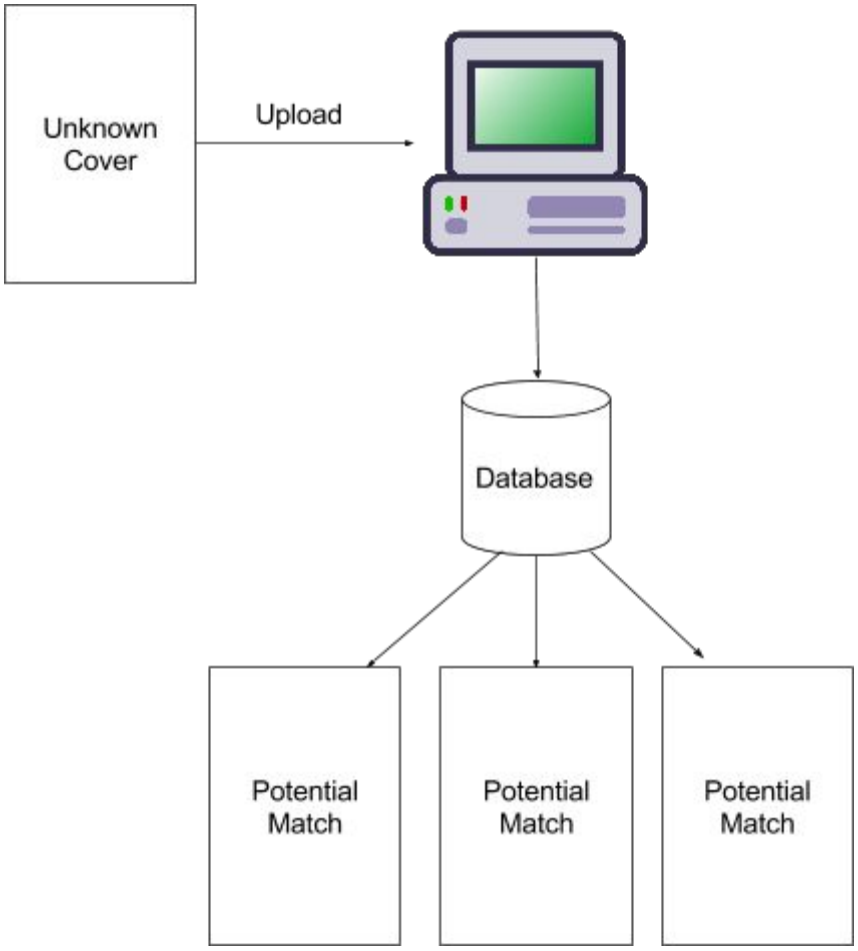

Figure 1

Database Population

Initially, the database was loaded with 200 images of known comic book covers. This gives us an initial database to test the searching against. A ColdFusion script was created that scrapes the ComicVine.com API [Comicvine.com 2015] and grabbed a random selection of comic books. A different ColdFusion script, combined with a bash script, was used to send a cURL [Curl.haxx.se 2015] request to the ComicVine API

and download the image URL that was returned from the API call. These images were saved using the ComicVine API ID for easier processing and future API calls.

Once all the images were downloaded, a stored procedure was created in Oracle that looped over the directory of images and added all of them to the database. During this insertion, we saved the ComicVine.com ID to the database for future use. We also saved the image both as a raw image in a BLOB and as the Oracle Still Image data type. This allows both the original raw image to be downloaded at a later time and also to run other Still Image functions against the image during later procedures. The code for the image insert is shown below.

```
create or replace PROCEDURE insertImage(imageName IN VARCHAR2) IS
        src_file bfile;
        dst_file BLOB DEFAULT NULL;
        lgh_file BINARY_INTEGER;
        l_coverId NUMBER;
        l_stillimage SI_StillImage;
        l_comicVineId number;
        matchCount number;
        l_avgcolor SI_AverageColor;
        l_colorhist SI_ColorHistogram;
        l_poscolor SI_PositionalColor;
        l_texture SI_Texture;
BEGIN
 l_comicVineId := SUBSTR(imageName,1,INSTR(imageName,'.'));
 SELECT
  COUNT(1)
 INTO
  matchCount
 FROM
  covers
 WHERE
  comicVineId = l_comicVineId;
 IF matchCount = 0 THEN
  INSERT INTO
   covers
  (
   rawImage,
   comicVineId
  ) VALUES (
   empty_blob(),
   SUBSTR(imageName,1,INSTR(imageName,'.'))
  ) RETURNING coverId INTO l_coverId;
  SELECT
   rawImage into dst_file
  FROM
   covers
  WHERE
   coverId = l_coverId;
  src_file := bfilename('COMIC_COVERS', imageName);
  dbms_lob.fileopen(src_file, dbms_lob.file_readonly);
```

```
    lgh_file := dbms_lob.getlength(src_file);
    dbms_lob.loadfromfile(dst_file, src_file, lgh_file);
    l_stillimage := new si_stillimage(dst_file);
    l_avgcolor := new si_averagecolor(l_stillimage);
    l_colorhist := new si_colorhistogram(l_stillimage);
    l_poscolor := new si_positionalcolor(l_stillimage);
    l_texture := new si_texture(l_stillimage);
    UPDATE
      covers
    SET
      rawImage = dst_file,
      si_image = l_stillimage,
      averageColor = l_avgcolor,
      colorHistogram = l_colorhist,
      positionalColor = l_poscolor,
      texture = l_texture
    WHERE
      coverId = l_coverId;
    dbms_lob.fileclose(src_file);
  END IF;
END insertImage;
```

Image Search Possibilities with Oracle

Once the database had been established, there were three different ways that the Still Image functionality in Oracle was used to search the database.

First, Oracle provides a Feature List function that takes in a Still Image and weights across 4 different criteria: average color, color histogram, positional color and texture, and generates a "Features List" object associated with the Still Image. Initially, we weighted all 4 criteria the same at 1, just to see how the entire process worked. Once a Feature List has been created, Oracle has a function to score an unknown Still Image based on the previously generated Feature List. In practice, this gave us 2 different options for searching the database: generate the feature list for the unknown image and then score each image in the database using this feature list. A second alternative is to generate a feature list for each image and scoring the unknown image based on the feature list of each image in the database.

As a third alternative, Oracle also provides scoring methods for each individual criteria of the image. We also attempted to write a different function that would score in the same way as the feature list above, but using the 4 individual criteria above. Below is a sample of the code for the function that searches the database for an unknown image.

```
create or replace PROCEDURE compare_images(
  imageName IN VARCHAR2,
  coverRefCursor OUT sys_refcursor
)
IS
```

```
        t_image   BLOB;
        l_stillimage SI_StillImage;
        l_avgcolor SI_AverageColor;
        l_colorhist SI_ColorHistogram;
        l_poscolor SI_PositionalColor;
        l_texture SI_Texture;
        l_featurelist SI_FeatureList;
        l_count INTEGER;
        l_coverId NUMBER;
BEGIN
        insertTestImage(imageName,l_coverId);
        SELECT
                si_image
        INTO
                l_stillimage
        FROM
                testCovers
        WHERE
                testCoverId = l_coverId;

        -- Generate a signature:
        -- l_avgcolor := new si_averagecolor(l_stillimage);

        OPEN coverRefCursor for SELECT
                coverId,
  comicVineId,
  SI_ScoreByAvgClr(averagecolor,l_stillimage) AS avgColorScore,
  SI_SCOREBYCLRHSTGR(colorhistogram,l_stillimage) AS colorHistScore,
  SI_SCOREBYPSTNLCLR(positionalcolor,l_stillimage) AS posColorScore,
  SI_SCOREBYTEXTURE(texture,l_stillimage) AS textureScore
 FROM
        covers c
 WHERE
  SI_ScoreByAvgClr(averagecolor,l_stillimage) < 10
  AND ORDSYS.SI_SCOREBYCLRHSTGR(colorhistogram,l_stillimage) < 10
  AND ORDSYS.SI_SCOREBYPSTNLCLR(positionalcolor,l_stillimage) < 10
  AND ORDSYS.SI_SCOREBYTEXTURE(texture,l_stillimage) < 10
 ORDER BY
  avgColorScore,
  coverId;
COMMIT;
END;
```

Experiments and Results

We conducted experiments on all three search methods. We evaluated each method on two criteria: accuracy and speed. Accuracy addresses how well the search function was able to find the exact same image in the database. Speed addresses how much time was required for a single image search.

Dynamically Computing Feature Lists - Weights Set to 1

In our initial experiment, a feature list was created on the unknown image and the score method was used on each row in the database. This search method had perfect accuracy, finding the unknown image image in the database. The method was able to successfully locate the matching image in 115 seconds.

Pre-Computed Feature Lists - Weights Set to 1

The next experiment used the same criteria (feature list with weights set to 1), but this time the feature list was generated and stored in a column in the database for each of the known images. The score method was then used with the feature lists from the database and the unknown image, which was again an exact copy of an image in the database. Again the procedure was able to successful match the image in the database and this procedure also took 115 seconds.

Individual Criteria Scores

In this experiment, we created a column for each of the 4 Still Image criteria in the database and populated it for each of our known images in the database. The score method for the individual criteria were used to score the unknown image against the known images in the database. When used individually, these methods would return a matching image in 25 seconds. Collectively, all 4 score methods could be run in 32 seconds, returning an exact match.

Image Distortions

There were also several experiments run on the initial feature list procedure to determine how the Oracle Still Image functionality would handle images that were distorted in some way. We used the same exact match image we used in previous experiments, but performed several image manipulations on it to distort the image. These experiments were not testing speed, but rather accuracy.

Blurry Effect

When a blurry effect was applied to the image, the correct image was still found in the result set and was the highest ranked result.

Black & White

When the black & white filter is applied to the unknown image, the result set that is returned is very small and the correct image is not found within the result set.

Darkened Image

When the image is darkened, a large result set (nearly 75% of all images) are returned, but the correct image is not found in the result set.

Lightened Image

When the image is lightened, a majority of the images in the database are returned in the result set. The correct image is found in the set, but is ranked very low.

Rotate 180^{O}

When the image is rotated 180 degrees, we receive a nearly identical result set to the control search. The correct image is found in the set and is near the top of the rankings.

Rotate 27^{O}

When the image is rotated 27 degrees, a very small result set is returned. The correct image is found in the result set, but it is ranked very low.

Database Size Increase

The final experiment increased the size of the database from 200 images to 1000 images. Here we were looking to see if there was a fixed start-up cost for running the Still Image functionality and if there would be a leveling off of the time it takes to run the query on a larger data set. When the original feature list procedure was run on the database with 1000 records, it successfully located the matching image in 50 seconds. This is compared to the result from the previous experiment where we achieved a speed of 25 seconds. As you can see, the search time doubled, while the database size increased by 4.

Evaluation

As an off-the-shelf solution, Oracle does not have the ability to do what we wanted it to do: create an indexed database of images that can be quickly and easily searched when given an unknown image. Oracle's strengths are the ability to store the images and to compare two images. However, when trying to use these capabilities to search, it is just too slow to return a result

Additionally, as evident in the results shown from our image distortion experiments, configuring the Oracle functions to match images when there is variance in the quality of the image would be difficult. It would be difficult to come up with a weighting scheme that would match all the possible distorted images that a user could provide. Even guiding the user to take the best picture possible, lighting and quality of the camera would greatly effect the results of the search.

Conclusion and Future Work

Overall, while it is nice to have some of the features, Oracle still isn't the perfect database solution for all image processing and handling needs. Throughout the entire project, we had difficulty finding information on how to do even the simplest tasks involving images in Oracle. Just the process of importing images into the database involves several settings, converting binaries to BLOBs, running functions to create Still Images, etc. All in all, this is not documented well and can be a headache for developers

In the future, there is much work that could be done to expand upon this project. First, if Oracle ever implements better handling for image searching, this project could easily be modified to take advantage of any changes they make. Second, this project could also be taken in a different direction by researching image comparison algorithms and applying this knowledge to the project. This would get away from the dependence on Oracle's tool set.

Bibliography

Comicvine.com,. 2015. Comic Reviews, News, and Forums - Comic Vine. http://www.comicvine.com/api/.

Cryer, J. 2015. Resemble.js : Image analysis. *Huddle.github.io*. http://huddle.github.io/Resemble.js/.

Curl.haxx.se,. 2015. cURL - How To Use. http://curl.haxx.se/docs/manpage.html.

Docs.oracle.com,. 2015. Oracle Multimedia SQL/MM Still Image Object Types. http://docs.oracle.com/database/121/AIVUG/ap_stimgref.htm.

Google Developers,. 2015. Custom Search | Google Developers. https://developers.google.com/custom-search/?hl=en.

Imageidentify.com,. 2015. The Wolfram Language Image Identification Project. https://www.imageidentify.com/.

Oracle.com,. 2015. Pre-Built Developer VMs for Oracle VM VirtualBox | Oracle Technology Network. http://www.oracle.com/technetwork/community/developer-vm/index.html.

Tineye.com,. 2015. TinEye Reverse Image Search. http://TinEye.com.