IMAGE COMPARISON VECTOR MODEL – FINAL PROJECT

Emily Cheng echeng6@jhu.edu

cs466 Information Retrieval and Web Agents

PROJECT DESCRIPTION

For my final project, I took what we learned from all four homework assignments and implemented an image comparison vector model, that uses a web robot to collect images from the internet, and decide if it matches a user uploaded image. In this case, a match is considered to be, by human evaluation, an image that two images that are essentially the same, but one may have been edited or cropped in some way (so "similar" images are not a match). The program itself consists of two parts, `main.prl` and `robot.prl`, which can simply be run with the command

```
perl main.prl
```

```
=========================================================
==     Welcome to the Image Comparison Web Robot       ==
=========================================================

Options:
    1 = Find matches to an image on the web
    2 = Use Image::Magick's to find matches to an image on the web
    3 = Run similarity tests for a pair of images
    4 = Run similarity tests for a pair of images using Image::Magick
    0 = Quit
```

There are four available options. Two of them send the robot off to traverse the internet (beginning at a default website or a user-specified one) and see if a user-specified image matches any images the robot pulls. The other two options simply internally run every comparison test available for two user-specified images, output the similarity scores to the screen, and then also tells the user if the images match or not, which is guaranteed if it passed any tests. These two options were used for most of the empirical testing that is summarized in results.pdf, and this data was used to determine which and how the comparisons would be run for the internet searching. As a control of sorts, I used the module `Image::Magick`'s compare method, which calculates the similarity between two images, and compared it against my own program. The user can upload an image from the current directory, a specified path, or use an image URL.

While the "big future" goal of this project is to detect art theft on the internet, the small scope of this project limited it to simple image comparisons within a very controlled environment. Evaluation was done by taking the cow image from this website, editing it in multiple ways, and then seeing if the program could classify the edited and original image as the same. Thus, the web robot portion of this project is very small, particularly since the robot itself is based off the HW4 robot, and crawls very slowly. The actual comparisons made allude to the classifications done in HW1, in which the first test it passes (or very much fails) automatically returns a result classifying it a certain way. Even though there are 17 distinct comparisons made,

they are all combined in such a way that there are only actually 5 specific tests the program can pass or fail. This decision is based off empirical data from a lot of testing, and is described more in detail in the "deciding if two images are the same or not" section.

The actual similarity and comparison functions mimic HW2 and HW3, using the very familiar vector model. Like the "bag of words" model, this program, for the most part, follows a "bag of colors" model where the color code of every pixel can be considered a distinct term.

SUMMARY OF TESTS

There are 4 main color tests, each which can be applied to either the whole image or segments of the image. Currently, for segmentation, the program is set up so that it splits, as best it can, the image into 25 equal pieces. Originally it had been 100 segments, but my results ended up showing that the difference between the two was miniscule, and some of the results by the 100 segments returned worse similarity scores. The program will then run each color test on all 25 segments and compute the average scores amongst those 25 segments. Thus, this program decides whether two images are similar or not based on at most 17 different tests.

SCALING TEST cmp_scale() (1 evaluation)

This test simply looks at the dimensions of each image and computes how similar the dimensions are between them. If the two have completely different dimensions, this is a good indicator of an image being completely unrelated, although potentially heavy cropping/scaling could have occurred.

COLOR TEST cmp_color() (6 evaluations)

This test is split up into 3 different parts, but the results of each are considered as one. Since every color can be split into RGB values, I compute scores for the red, green, and blue channels. This test runs by first making a histogram of color values for each channel for the given image (or segment), seen in the `make_color_hist()` subroutine. It will linearly look at every single pixel on the image by using `Image::Magick`'s `get('pixel[x,y]')` function, which returns a string of its hex color codes. The string follows the following format:

```
#####,#####,##### = REDVAL,GREENVAL,BLUEVAL
```

This string is then split on the commas, and each color value is stored to its respective channel. These values represent 16-bit color codes, accounting for 65,536 colors for each channel, so I reduce them to 8-bit (255 colors) to simplify it (otherwise even the slightest variations would result in huge differences). Each histogram is stored as a hash of specific color code key to the number of times it appears, which is then normalized to represent a percentage of the whole image that color is in. I did not normalize the colors based on how many times the color appears in the whole image (compared to TF-IDF weighting in HW2 and HW3), because the number of times a specific color appears is very important in regards to an image.

Once all histograms are made for both images, it uses the cosine similarity function to compute the similarity between each color channel. The color codes are like terms in a document, and is treated as a "bag of words" (or "bag of colors" in this case). The function then

returns these results. This test is ideal for instances of when the image has been dimensionally altered, like flipping or light cropping, and thus works better when applied to the whole image. For segments, due to instances of cropping and orientation, color distributions can completely leave one segment and enter another, disrupting the similarity between two paired up segments. Thus, I only run this test for the whole image.

HUE TEST cmp_hue() (6 evaluations)

This test is similar to the color test in that it also looks at each of the 3 color channels using the generated histograms. However, it then takes the color distributions (weight) and takes the difference between each image by first organizing the hashes from largest distributions to lowest, and then subtracting the distributions between the two. If two images have similar color distributions, this score should be very close to 0 (the histograms "cancel each other out"). Thus, this test then calculates a score by seeing how many of the colors canceled out evenly to have a score of less than 0.1. Thus, this test is sensitive to when an image has been altered via the color channels, like applying a filter, increasing saturation, changing hue, etc.

Since the program treats the colors as a "bag of colors", this score almost always returns high when applied to the whole image, regardless of the actual similarity between the two images. However, this test is much more accurate and useful when evaluating segments. Thus, I only run this test on the segmented images.

GRAYSCALE TEST cmp_gray()

This test works similarly to the color test, except it first converts each image to greyscale using `Image::Magick`'s applications, and thus only creates one histogram of gray values (since for a gray color, the color codes for RGB will be the same). It then performs the cosine similarity between the two grayscale histograms. This test has shown to be very weak overall, and is thus only used in context with the black and white test.

BLACK WHITE TEST cmp_bw()

This test converts both images completely into strictly black and white pixels. It then runs through the entire image and looks at the *positions* of each pixel. If the pixel at position (x,y) on the first image matches the color of the pixel on the second image at position (x, y), the pixels "match", and this program calculates how many of these pixels end up matching overall. Thus, this program is sensitive to the position of dark/light areas.

This test is very binary in that it either scores very high or very low. This is because when colors have been edited, which pixels turn black and which pixels turn white varies greatly. Similarly, if an image has been cropped, the locations of these pixels will have shifted, although this program attempts to normalize the positions by noting the offset between the larger and smaller images, and centering it. Thus, this program is only runs in context with the grayscale image for segmented images.

DECIDING IF TWO IMAGES ARE THE SAME OR NOT

I collected empirical data by taking one image, cow2.jpg, and editing it in different ways. I then compared the original image with the edited images by running every possible test between the two images. The final data is shown in results.pdf, which also includes comparisons between cow2.jpg and completely unrelated, but perhaps visually similar, images (some of these found by using Google's reverse image search function). An image "matched" if it passed any of these tests. It is this data that I used to decided how my program would evaluate the similarity between two images. The images used are stored in the folder labeled "images". My other data is stored in the "misc" folder, which is what I stored as I was still building my program. It is currently incredibly unorganized, and difficult to understand as I changed many calculations throughout writing this program, but it is there for reference.

The program first looks at the scale between the two images. If the two images are less than 80% similar in terms of scale, then it is immediately classified as not the same, thus filtering out a large majority of grabbed images at the expense of not correctly classifying very few instances of the image being extremely cropped. It then runs the color test for the whole image. If the combined similarity scores of each color channel is above 2.70 (alluding to over 90% similarity in each color channel), then the program immediately classifies it as a match.

If not, the program then segments the images. It then runs the hue test on these segments. Similarly, if the collective hue distributions combined score above 2.7, then the program classifies it as a match. If the collective hue distributions combined scores lower than 1, it immediately classifies it as an unrelated image. If two of the channels combined score over 1.8 (.9 for each channel), this implies that filtering/hue adjustment through one channel was applied, which would result in a low score for one of the channels but a high score in the other two, so the program classifies this as a match.

Finally, the program looks at the grayscale and black/white distributions of the segments. If the gray test scores over 60% in similarity AND the black and white test scores above 75%, the program classifies this as a match. Truthfully, I don't believe any image I tested passes this test, but I did not want to lower the threshold or apply it to the whole image, since applying it to the whole image would allow many grayscale images to pass the test.

The reason I opted to only run a few actual tests despite the numerous combinations of tests is because, as I learned through a lot of trials in this assignment, is that each image is extremely sensitive to specific things based on how it was edited. Light cropping and rotation does not change the actual color distribution nor scale of the pixels much, which is why it relies foremost on the color test. However, applying filters and enhancing the image are much more dependent on the segment hue and grayscale/black/white tests, which pay more attention to light and intensity. My classification does, however, begin to run into issues once multiple enhancements are combined. However, being selective with these measurements also ensured that completely unrelated images would not be classified as a match, which is a problem that `Image::Magick` had.

`Image::Magick` had its own `compare()` function, which would compute the similarity between two images. Based off some empirical tests, I decided to classify that an image would be classified as a match if the Magick score was over 75% in similarity. The issue I found with Magick is that it would return high results for almost any image, including completely unrelated images. I'm not exactly sure how Magick calculates similarity, but while it is more likely to match extremely edited images, it is also much more likely to pull completely unrelated images.

For example, when sending off the robot to look for related images to cow2.jpg starting at http://www.cs.jhu.edu/~jkloss/self/index.html, which is where I got the cow.jpg image, Magick would return the cow image as well as a multitude of unrelated gray images:

```
Looking at http://www.cs.jhu.edu/~jkloss/self/index.html
    Image match: http://www.cs.jhu.edu/~jkloss/images/grey_self.jpg
    Image match: http://www.cs.jhu.edu/~jkloss/images/cow2.jpg
Looking at http://www.cs.jhu.edu/~jkloss/rowing/index.html
    Image match: http://www.cs.jhu.edu/~jkloss/images/grey_rowing.jpg
Looking at http://www.cs.jhu.edu/~jkloss/research/index.html
    Image match: http://www.cs.jhu.edu/~jkloss/images/grey_research.jpg
Error: 404 Not Found
Looking at http://www.cs.jhu.edu/~jkloss/java121/index.html
    Image match: http://www.cs.jhu.edu/~jkloss/images/grey_java121.jpg
Looking at http://www.cs.jhu.edu/~jkloss/self/index.html
```

cow2.jpg matched itself, as well as the following:

When my own comparison calculations ran against the same website, it would only return the cow image. Similar results are seen in results.pdf, where Magick matched other completely unrelated images. My program, on the other hand, was at the very least consistent in that it would not match a completely unrelated image. One I idea I played around with only briefly was incorporating both the Magick score and my own when classifying an image as a match or not. However, since I am unsure on what basis Magick makes its score, and since Magick proved overall more unfavorable, I did not heavily pursue this direction.

PROBLEMS AND ISSUES

As noted earlier, my program did not match images that had been manipulated too much, like some combination of dimensional and color manipulation, even if I can also say that Magick did not match it very well either. Looking at the data, even if I had used every single possible test to classify an image, it would have still been difficult to classify an overly-edited image as a match since it tended to score low for everything regardless. Some other tests I could have made could have edited the original uploaded image and see if the compared to image had been manipulated that specific way, but doing such makes my program too specific and not a very efficient "catch all" program. Furthermore, since my program, combined with a slow web crawler, is already slow and a bit inefficient (calculating all those histograms, the BW test is especially expensive since it linearly looks at the position of all the pixels), adding many more manipulation tests would have slowed it down even more.

As far as the web robot goes, I noted this issue in HW4 but was not able to resolve it. For some reason, when I run from the undergraduate servers, I am not able to access secured websites (like https://.... versus plain http://...). For example, if I attempted to run the robot on www.cs.jhu.edu, the robot returned `Error: 501 Protocol scheme 'https' is not supported (LWP::Protocol::https not installed)` and did not go anywhere, so the program ended very quickly. I was able to do some minor tests by running it on a different machine, but since the tests evolved and I kept adding new test data, I was not able to effectively test the web robot part of my project. Thus, I worked on the assumption that it works similar to my HW4 robot, and where I was able to run it, the program returned expected results.

I also worked mainly with JPGs and PNGs, but assuming `Image::Magick` works the way it is expected, the format should not matter. I did run into some sort of IO error with GIFs though, which appeared to be some internal issue, so I had to resolve it by ignoring them.

OVERALL EVALUATION

Because there are so many factors involving image comparison, I had to be very careful of choosing tests that were as general as possible but still covered a range of situations. I realize that I worked within a very limited and controlled environment, but I feel that the results that I did get for this limited range are still informative. The complexity of the program isn't extremely high, but unlike words in which one can consider context, it's position relative to other words, and frequency, every single color evaluated in an image is important. Positions of the colors are important as well in the sense that colors can be found anywhere (unlike sentence structures would can expect certain words/phrases to follow others), but where it ends up being is important, hence the introduction of segmenting the images.

Again, I was not sure which tests to implement and add without making the program look for very specific things and making the efficiency of the program worse than it already is. Furthermore, the spirit behind this project was for art theft, and in my experience, most images that are stolen are not edited much outside of slight cropping (to perhaps remove a watermark),

and maybe slight filtering. For the most part, art is stolen by simply being downloaded from one source and uploaded to another.