Content Based Image Retrieval

CAP5415 - Computer Vision Project

Adam Stallard
Florida State University
aps10d@my.fsu.edu

Eric Serbousek
Florida State University
els16@my.fsu.edu

1. INTRODUCTION

A fundamental problem in computer vision is being able to compare images. In this project, we have implemented three versions of image retrieval systems. The first two, are simple CBIR systems which compare images using histograms. The third method uses SIFT features, and a FLANN based matching algorithm.

This experiment was ran on two separate Ubuntu machines. One was virtualized with 8G of memory at 1333MHz and 4 dedicated CPU cores maxing at 3.4GHz. The other was non-virtual with 16G of memory at 1600MHz and 8 CPU cores maxing at 3.6GHz.

2. DEPENDENCIES

Python 2.7 was as chosen as the language, largely due to library support. Our code uses numpy 1.11.2, OpenCV 3.1.0-dev, matplotlib 1.5.3, and several standard libraries.

While most of the code was written to be version agnostic, the OpenCV version is critical. Since SIFT is a patented algorithm (Lowe), it is not free for commercial use, and is not available in the standard versions of OpenCV. The dev version must be compiled from source, using opencv_contrib libraries.

3. DATA SET

The test data set is that of 1000 images, ordered by file name "n.jpg" where n is the nth image in the data set. Every adjacent 100 images belong to their own category. Each category is a unique collection of similar images, where similarity may be defined by scene, objects, theme, etc.

Our system includes a method for downloading the data set and unzipping it into a named folder. At the start of run-time, all images are loaded into memory at once. In a real world scenario, this would not be plausible, and a database should be used. However, for the purposes of this experiment, this allowed for much better performance and fast test cycles. The total data set size is about 30M.

4. ALGORITHMS

The relevant algorithms used in this project include calculating precision and recall, histogram intersection, Gaussian blurring, bilateral blurring, extracting SIFT features, and approximate nearest neighbor searching via FLANN. Precision, recall were all done manually, while Numpy and OpenCV libraries were used to perform the rest of these operations.

Given a query image, our system tests this image against every other image in the database. Note that the query images are also from the database, and thus belong to 1 of the 10 categories. The ultimate goal is to correctly match this query image against images in its category.

Precision and recall values are calculated using the following:

$$\begin{aligned} P &= \frac{n \cap m}{m} \\ R &= \frac{n \cap m}{n} \end{aligned} \tag{1}$$

where n is the number of relevant documents, and m is the number of matched images retrieved. Given a query image, we plot for each test image the precision recall for $k \in (1, n)$. This is a common metric to measure retrieval system performance. As k gets larger, i.e. more images are tested against, the precision and recall values converge at a final point. To pick out good and bad representations from each category, we chose query images that generated minimum and maximum values of $P * R$ for $k == n$. Given that an optimal system will have perfect recall and perfect precision, thus maximizing this product, we decided this metric was reasonable. Tables 1, 2, 3 shows examples for categories 1, 5, and 9 for each of the three methods. Note that the bad examples show very low precision values. This shows that these query images were not good representations of their own category, and thus matched many images from other categories.

Our system generates a final PR plot for each category and method using all final P, R values from each query image (where $k = n$). For this experiment, we decided to run every image in the database as a query image. This means we made $3 * 1000 * 1000 = 3,000,000$ image comparisons. The virtual host was able to achieve this in about 40 minutes, where the other completed within roughly 30 minutes.

Table 1: Individual Precision-Recall plots for standard color histogram matching

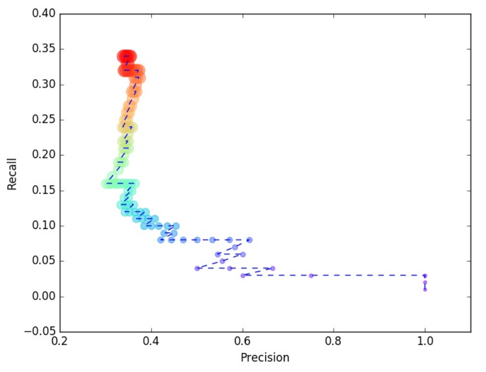
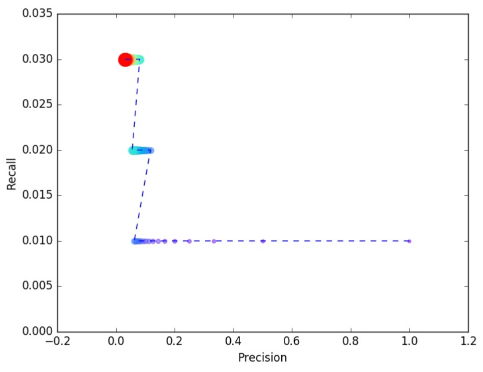
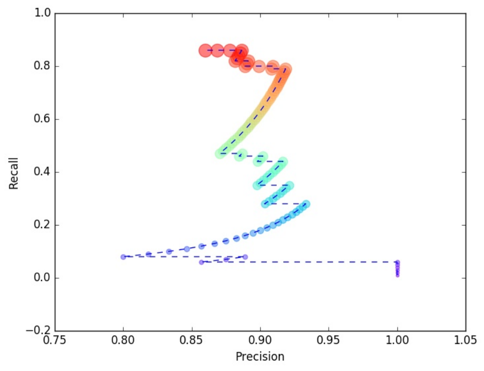
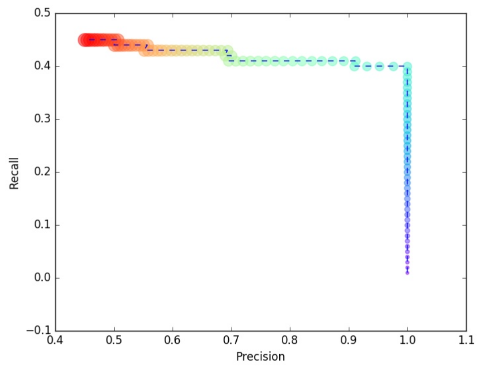
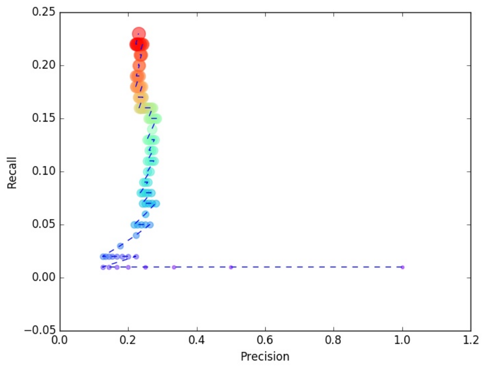
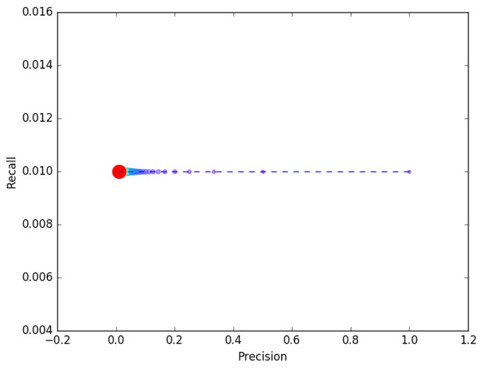
Category	Good examples	Bad examples
1		
5		
9		

Table 2: Individual Precision-Recall plots for filtered color histogram matching

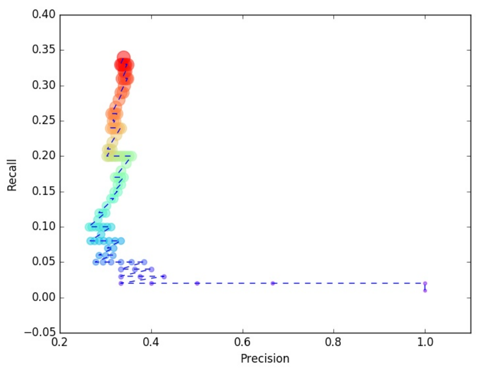
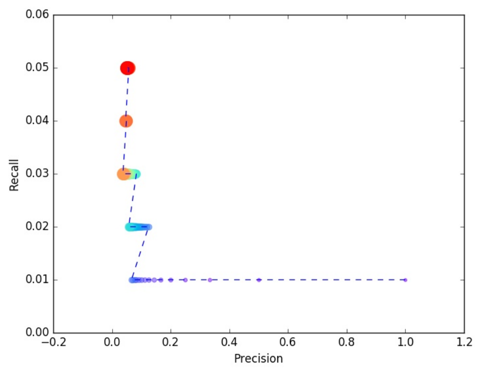
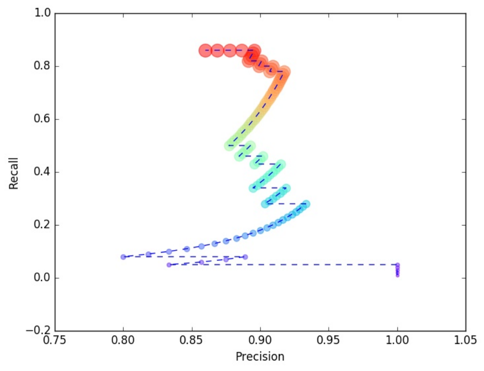
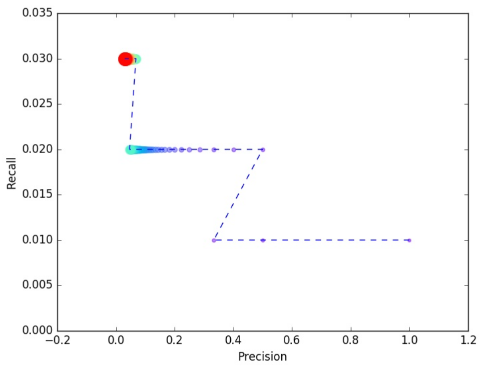
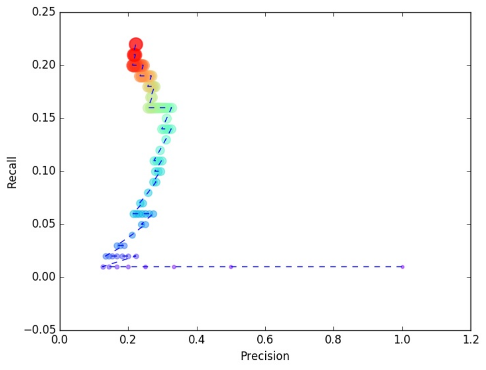
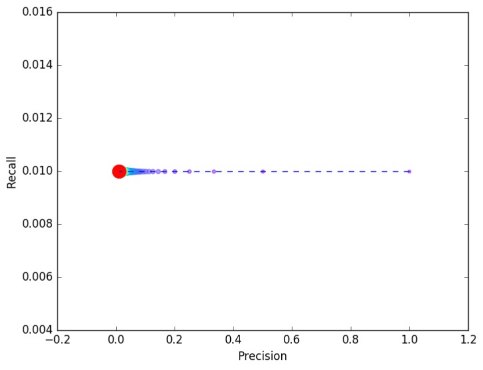
Category	Good examples	Bad examples
1		
5		
9		

Table 3: Individual Precision-Recall plots for FLANN based matching using SIFT

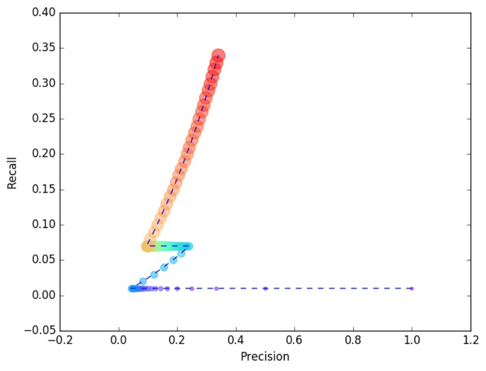
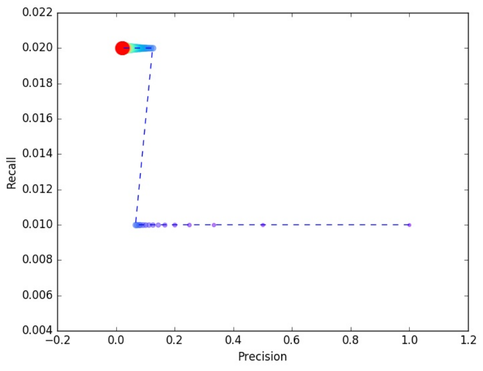
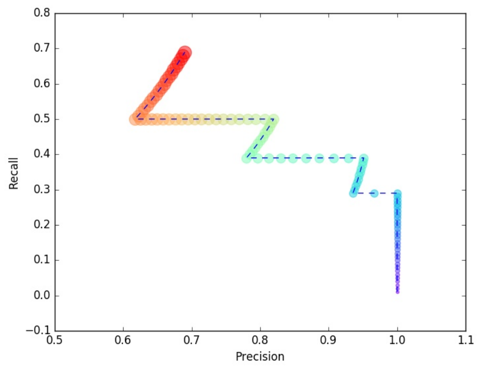
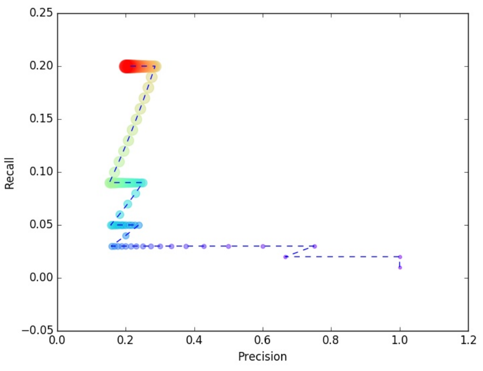
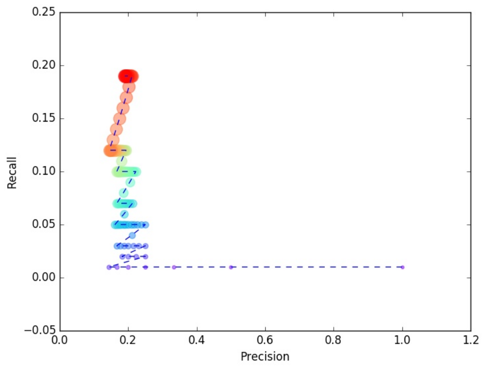
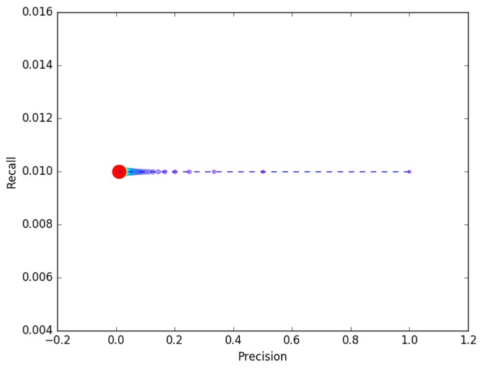
Category	Good examples	Bad examples
1		
5		
9		

Figure 1: Average precision. (Ro : hist, G+ : filtered, B* : SIFT)

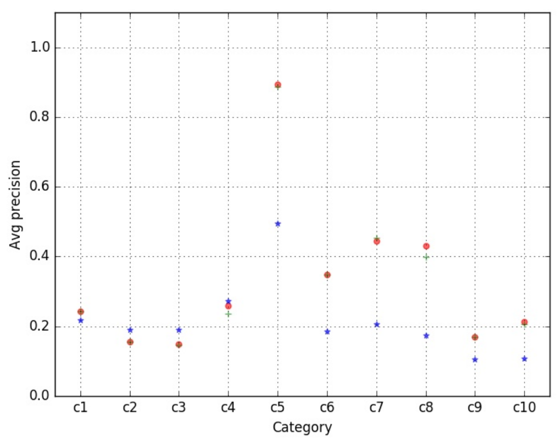
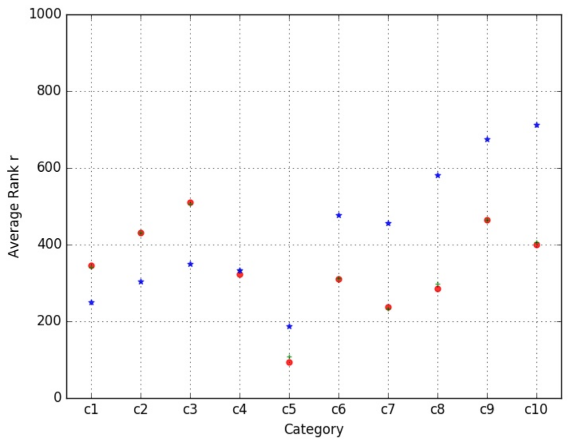


Figure 2: Average rank. (Ro : hist, G+ : filtered, B* : SIFT)



These time differences are likely due to the memory refresh rates, and the overhead of virtualization. Doing this allowed us to see a comparison of how each image in each category performed relatively. Table 4 show these full plots for categories 1, 5, and 9 for each matching method. This is a representation of how well each category is self related. That is, better categories have images that are more similar to each other than in other categories.

The final demonstration of precision and recall in this experiment is done by averaging the results for each category. Each of our methods maintains a set of precision recall results, and thus at the end of the experiment, each PR result from the query images are averaged by weight according to the following formula:

$$\frac{1}{n}\sum_{k=1}^{n}\frac{(n \cap m)^k}{k} \tag{2}$$

Here, k is the number of images retrieved, and $n \cap m$ is the number correctly matched out of k (the intersection of relevant images and matched images). These weighted precisions are then averaged for every query image in a specific class. The final averaged precision values for each category are shown in table 4.

Finally, for each method, we generate rank plots to see how well the query images match to each category, by measuring how "close" they are. That is, each test generates a metric of closeness between the query images and corresponding test images from the database. This metric is later used to determine how well the retrieved images that were true matches (n), compared to the query image. This is done by sorting the results by the metric (the match value), thus giving each result a "rank", and then filtering to yield only the relevant n images and their corresponding ranks. The ranks are averaged for each query image. In figure 4, these average ranks are plotted for each class, for each method.

Note that the ideal CBIR system will generate an average precision of 1, and average rank of 50. The two histogram matching methods are very similar with some slight variation. It seems certain categories respond differently to various filters. While most of these categories seemed to show good recall, there were a large number of false positives when matching strictly on histograms. This can be seen in the rank plot where ranks are much larger than 50. This means that out of all the matches returned, many of the correct ones had a smaller value for "closeness" as defined by the metric.

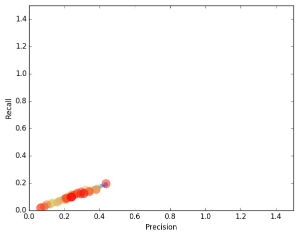
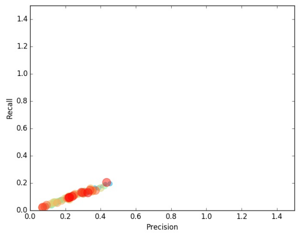
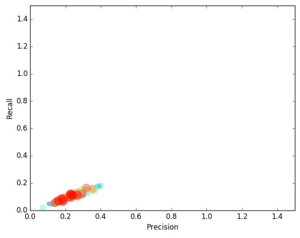
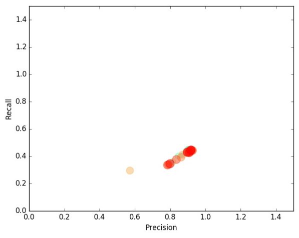
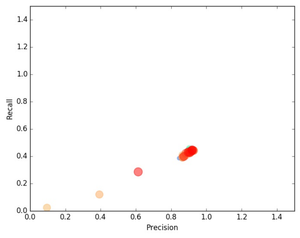
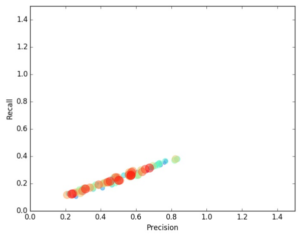
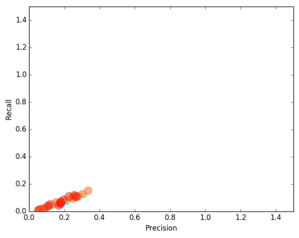
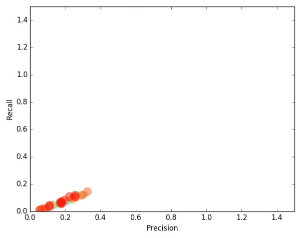
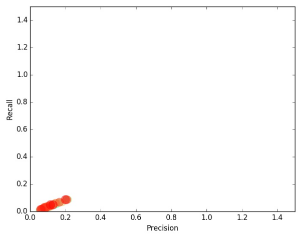
For SIFT feature based matching, the results come from OpenCV's FLANN matcher which essentially uses a KD tree to perform fast approximate nearest neighbor searching. As per Lowe's paper on using Scale Invariant Feature Transforms, a simple ratio test between matched pairs extracted from the FLANN matcher allows us to filter out many false positive matches. This improves the results of SIFT based matching.

It should be noted however that this requires a heavier load on the system. Histograms are relatively easier to compute, where as image descriptors are more cumbersome. For this reason, we chose a low number of descriptors to train each tree with, specifically 50. If a database was used, these descriptor sets could be computed one time, and then imported during each test cycle. However for the purpose of this experiment and given that the test data set was relatively small, this was not necessary. Also, the majority of the work for this matching method was done by the nearest neighbor search algorithm via FLANN trees.

5. CONCLUSION

For the final results, it seems that SIFT and histogram matching performed better in different categories. The ranks for SIFT matching were best in categories 1, 2, and 3. Histogram matching performed better in categories 5 through 10. Category 4 seems to show roughly equivalent results for each method. Given that the performance of the SIFT matching method is much worse, this suggests that histogram matching might actually be more efficient method. Perhaps if both methods were used in conjunction, and a linear summation with weights for each metric was used,

Table 4: Full Precision-Recall plots for k = n = 100 for each query

Category	Standard image histograms	Filtered histograms	FLANN via SIFT
1			
5			
9			

a more complete and effective matching system could be achieved. Given time constraints this was not implemented here, but may be an interesting topic for future work.

In a real world setting with much larger data sets, image descriptors would certainly need to be calculated and stored in a database for faster loading time. Several FLANN objects could be loaded into memory for each test image, and used in parallel when matching images. This would improve performance in a linear fashion. Of course, it does not improve the complexity of the algorithm, and thus may not be ideal for very large data sets. It may be possible to store a subset of match results between image pairs in various categories, providing a means of performing matches against a smaller set of test images, and possibly even a subset of categories. While a proper precision recall would not be generated, its practicality could be useful for large data sets.

6. REFERENCES

[1] http://docs.opencv.org/2.4/doc/tutorials/features2d/feature_flann_matcher/feature_flann_matcher.html

[2] https://www.scivision.co/compiling-opencv3-with-extra-contributed-modules/