Identifying Products in the Real World

Kevin Laube
kevlaube@stanford.edu

Dave Dolben
ddolben@stanford.edu

Oliver Toole
toolebox@stanford.edu

CS 231A Final Project
March 19, 2014

Abstract

The goal of this project is to successfully identify and locate products in photos by using a classifier trained on images taken from the web. Additionally, the goal was for the process to be mostly unsupervised; aside from setting up the data sets our algorithm had no prior knowledge about the products in the images. To achieve this, we constructed training sets consisting of multiple products. For each product in the training set, we included a varying number of images of the product taken from different viewpoints. We then used SIFT features[1] to construct a set of frequently occurring features for each product. For classification of new images, we compared each SIFT descriptor in the input image to our trained classifier and used a best matching algorithm to make a prediction.

Our algorithm was very successful on smaller image sets, and retained good accuracy as the size of the set grew. We felt this small loss in accuracy was reasonable, given the limitations of SIFT descriptors and training sets we could pull from the internet.

1. Introduction

Product identification offers many unique challenges in the field of computer vision. Typical methods include sliding window, linear classifiers, and other expensive methods. We propose a method in which SIFT descriptors for each product are collected from a large set of training images, and used to locate and classify a product in a query image.

Our method is designed to be trained in an unsupervised manner on a set of images of a product. Ideally, the training set would contain many images of the product from different angles, so that we can both capture many views of the product and give our training algorithm multiple images to use to identify the object in question.

Each training set we used consisted of a number of categories, each labeled with a product (e.g. Coca-Cola can). Each category had a number of images, similar across each category. The goal was to find a relatively diverse set of products/images while avoiding having two sets be overly similar. We also created multiple data sets of different sizes, both number of categories and number of images per category.

The final goal of the system is to take in an input image and assign it a label corresponding to the most similar category from the training set. In this way, we will identify the product that is in the input images.

2.1 Review of Previous Work

In the field of object identification and classification, many new and exciting algorithms attempt to increase accuracy, while decreasing computational complexity. The tradeoff between these two goals creates an array of methods, which lend themselves nicely to different types of problems. In our case, we are attempting to locate and classify products in a query image.

One of the most accurate object identification algorithms is the sliding window method. A a heat map is generated, indicating where the object is likely to be located. This heat map is very expensive, with computational complexity dependent on the size of the image, the scale and angle of the object, and the number of classes. Another shortfall is that many products in the real world are not rectangular in shape, which reduces the accuracy of this method significantly.

The bag of words representation of an image reduces computation by first clustering key points, which can then be used in a classifier such as an SVM. The major shortfall of this method is that locality of the object is completely lost. We would also like a method for which the location of the object in the query image is recoverable.

2.2 Methodology

Our method's advantages over those stated in the previous section have already been mentioned. The sliding-window/heatmap approach is expensive in both training and testing, and while it might provide higher accuracy, would not be suitable for fast lookup in a potentially real-time scenario such as video.

The main shortcoming with the bag-of-words approach is that it abandons orientation and location information about the object in question. This makes it impossible given the results to locate the object in the test image. Also, it requires the creation of a large dictionary of codewords via clustering, which would require more fine tuning to do in an unsupervised manner than our method.

3.1 Technical Summary

Before breaking down our technical approach to the problem, we will describe it at a high level. We broke the problem of product detection into two main subproblems:

1. Deciding how to train a classifer from a set of training images.

2. Picking good metrics for making a prediction given a new input image.

Our initial plan was to train a classifier corresponding to every label in our training set, and make predictions based on minimizing the prediction value across all classifiers on a new input image. While this method showed some promise, it didn't result in the high accuracy we were hoping for.

Our next plan was then to train one big classifier, containing descriptor data from all labels in our training set. With this new classifier, our prediction scheme was to match each descriptor in the input image with its best match in the classifier set, and make a prediction based on which label received the most votes in this manner.

This second method proved much more successful, and we were able to achieve very high success rates on smaller testing sets. However, scalability was a bit of an issue with larger data sets. The reasoning behind this decreased accuracy will be discussed later.

3.2.1 Initial Approach

First, we would like to go into some details about our first classifier model and explain how its fallacies lead to the intuition behind a better model. The training scheme behind the classifiers worked as follows. For a given data set, we would have K different object labels, each of which came with N images. For a given label i, classifier K_i would be trained by comparing all N images corresponding to label i against each other, for a total of $\frac{N*(N-1)}{2}$ comparisons. For each comparison, we would first extract the SIFT keypoints and descriptors of each image. We would then run the SIFT algorithm to remove keypoints that didn't pass the SIFT ratio test, and eliminate remaining outliers via RANSAC. At this point, we would be left with sets $A = (a_1, a_2, ...a_n)$ and $B = (b_1, b_2, ..., b_n)$ of keypoints in the two images being compared. For each of these keypoints, we would then give it a vote. After all $\frac{N*(N-1)}{2}$ comparisons, we would then have a histogram for each image, showing how many votes each keypoint in the image received during training. We would then keep a subset of these descriptors for classifier K_i, corresponding to the descriptors with the most votes.

After all K classifiers were trained, the model was ready to make predictions. For a prediction, the SIFT keypoints and descriptors were extracted from the input image Q. Then the set of descriptors from Q were compared to each of the K classifiers. For each descriptor d_q in Q, (where $q \in \{1...S\}$ and S is the number of descriptors in Q), the algorithm would find the descriptor $dopt_{q,K_i}$ in classifier K_i that was a minimal distance from d_q (measurement was based on L2 norm). The prediction value for classifier K_i would then be $P_i = \sum_{q=1}^{S} ||dopt_{q,K_i} - d_q||_2$. The final prediction for the image was then the label corresponding to the classifier with the minimum value for P_i.

Unfortunately, this method was only marginally successful. It was able to achieve 70% accuracy on a smaller testing set but was unable to do better then 40% on the bigger sets. The biggest issue we saw was that the prediction values P_i were very similar. This pointed to a shortcoming with the sum of norms method that we were going for; it failed to produce enough variance between similar keypoints and very different keypoints.

3.2.2 Single Set Classifier

Our final algorithm design was to instead train one classifier set, consisting of (descriptor, label) pairs drawn from all images in our training set. The new training scheme was the same as the previous classifier, but now all descriptors remaining after SIFT and RANSAC were added to one single set, to make our classifier C. One issue we were concerned with was varying number of descriptors from different labels. This was in part due to different image sizes, but also due to the lack of consistency in keypoints that pass the SIFT and RANSAC metrics. To control for these, we would allow at most ND descriptors per image label in C. Thus, C would contain at most $ND * K$ elements, where K is the number of labels in the training set.

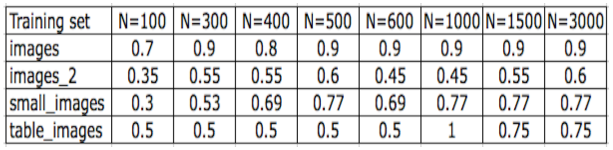
For our new classification scheme, we would begin with an input image Q_i and compute its SIFT keypoints and descriptors. The program then initializes a histogram H_i with K buckets, initially all set to 0. Next, the program iterates through every descriptor d_j in Q_i and finds the descriptor $c_{j,opt}$ in C that minimizes $||c_{j,opt} - d_j||_2$. Considering the large size of C, this lookup step had to be very efficient. To facilitate this, we used a k-d tree structure to organize the descriptors in C. Once we found $c_{j,opt}$, we also knew the label corresponding to $c_{j,opt}$. This label would correspond to an index of H_i, and we would then increment the appropriate bucket by 1. Thus, the prediction on image i would be the bucket of maximum size in H_i after all descriptors were examined.

3.2.3 Optimizing Classifier

Our new classifier initially performed at least as well as the old classifiers on all datasets, and even outperformed them in some instances. However, there was clearly a lot of tweaking to be done. The biggest issue was the variable ND, which controlled how many descriptors we allowed in our final classifier. The variance in number of descriptors obtained after SIFT and RANSAC ran across different labels was huge; some labels produced a couple hundred descriptors while others produced thousands. This lead to the idea of setting a ceiling on the number of descriptors we would allow in the final classifier from any one label. Different values of ND produced very different results as well; too few and the model lost too much information from training, too many and images with more descriptors from training would tend to dominate the predictions. This indicated the optimal value for ND was a function of the number of images per label, and to test this theory we ran tests on multiple data sets with different values for ND. These tests were done on deterministic training and test sets, so there was no random elements. Figure 1 below shows the results obtained.

Based on the results from testing, we set different values for ND based on the number of images per label in each data set.

Additionally, we tried a few different schemes for summing up histogram votes. Initially, we would just increment each bin of the histogram by 1 every time the given



Training set	N=100	N=300	N=400	N=500	N=600	N=1000	N=1500	N=3000
images	0.7	0.9	0.8	0.9	0.9	0.9	0.9	0.9
images_2	0.35	0.55	0.55	0.6	0.45	0.45	0.55	0.6
small_images	0.3	0.53	0.69	0.77	0.69	0.77	0.77	0.77
table_images	0.5	0.5	0.5	0.5	0.5	1	0.75	0.75

Figure 1: Different accuracies achieved with different numbers of descriptors. Number of images per label in each set increased from top to bottom.

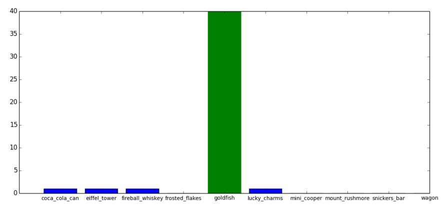


Figure 2: Example of the histogram showing a correctly labeled goldish product.

index got a vote. Our next idea was to apply the SIFT ratio test to filter out less likely matches. To do this, we compared the labels of the 2 nearest neighbors found for each descriptor d after the k-d tree search, m_1 and m_2. If $||d - m_1||_2 < .75 * ||d - m_2||_2$, we would conclude m_1 was a suitable match and increment the appropriate bin in the histogram. Otherwise, we would throw out the vote. This method improved our results substantially.

Additionally, we considered scaling the value we would add to the histogram. The intuition behind this was that certain descriptors would be 'more indicative' than others. As an example, consider a prominent descriptor on a Coca-Cola can. Intuitively, it seems that this descriptor would receive many votes when training the classifier, and would be more indicative then others of a product being Coca-Cola. Thus, if this descriptor were matched during training, we would increment the appropriate bin of the histogram by a larger value corresponding to the number of times it remained after SIFT and RANSAC. This optimization didn't prove very effective, but it didn't hurt performance and did help in some cases, so we decided to keep it.

3.2.4 GrabCut Segmentation

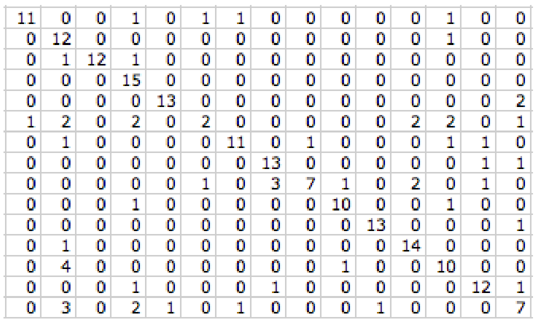
Once we have the final set of query image descriptors, we can now identify the object in the scene by using the GrabCut library from openCV. The idea is that after SIFT and RANSAC find the best matches in the query image, only those keypoints lying on the object contribute to its classification. From there, we can estimate a bounding box for the object based on the distribution of the keypoints. Lets call the set of valid keypoints in the query image S. In our system, we calculate the centroid of the points in S and call that c. We then calculate the width and height of the semiaxes of a bounding ellipsoid of the keypoints, h, l. We pass in the ellipsoid defined these parameters (c+h/2,c-w/2), (c+h/2,c+w/2), (c-h/2,c-h/2), (c-h/2,c+h/2) into the GrabCut function, as an estimate of the object position. With some certainty, we can claim that the object roughly lies inside the ellipsoid, and the background of the image is roughly everything outside of the ellipsoid.

For correctly classified objects, this method was suprisingly succesful. Complete object segmentation occured in many cases, specifically when the keypoints distributed evenly around the object.

4 Experiments

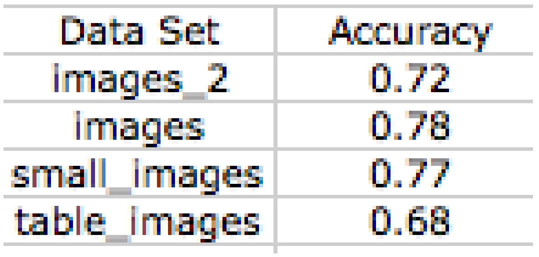
After we ran a sufficient amount of deterministic tests to tune our model, we ran more exhaustive, non-deterministic tests on several data sets. As a proof-of-concept, we wanted to verify that we would get around 100% accuracy when our training data set included images from the test set. Unsurprisingly, we had 100% accuracy on all sets with this testing methodology.

Next we began the real testing phase. For these tests, we removed one image from the image sets corresponding to each label, and put these images in a testing set. The remainder of the images were placed in a training set, and a model was trained on this set. Finally, we fed the testing images we had set aside back in to the classifier to see the prediction results. Some figures and explanations are given below.



11	0	0	1	0	1	1	0	0	0	0	0	1	0	0
0	12	0	0	0	0	0	0	0	0	0	0	1	0	0
0	1	12	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	15	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	13	0	0	0	0	0	0	0	0	0	2
1	2	0	2	0	2	0	0	0	0	0	2	2	0	1
0	1	0	0	0	0	11	0	1	0	0	0	1	1	0
0	0	0	0	0	0	0	13	0	0	0	0	0	1	1
0	0	0	0	0	1	0	3	7	1	0	2	0	1	0
0	0	0	1	0	0	0	0	0	10	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	13	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0	14	0	0	0
0	4	0	0	0	0	0	0	0	1	0	0	10	0	0
0	0	0	1	0	0	0	1	0	0	0	0	0	12	1
0	3	0	2	1	0	1	0	0	0	1	0	0	0	7

Figure 3: The confusion matrix above shows the distribution of correct and incorrect guesses after testing.



Data Set	Accuracy
images_2	0.72
images	0.78
small_images	0.77
table_images	0.68

Figure 4: The results above were obtained via non-deterministic testing on randomized training/testing images

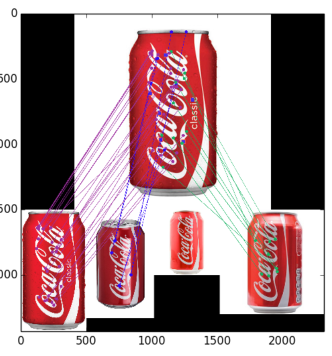


Figure 5: Keypoints of test image matched with training images of a Coca-Cola can.

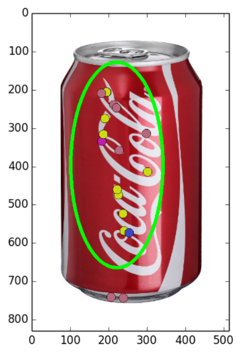


Figure 6: Keypoints and estimated bounding ellipse for Coca-Cola can.

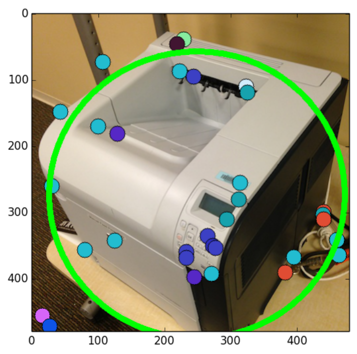


Figure 7: Keypoints and estimated bounding ellipse for HP Printer.

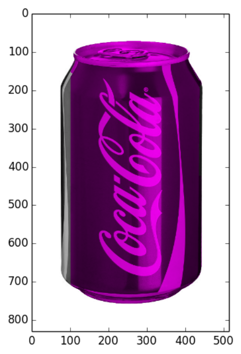


Figure 8: Final results showing segmented image for Coca-Cola can.

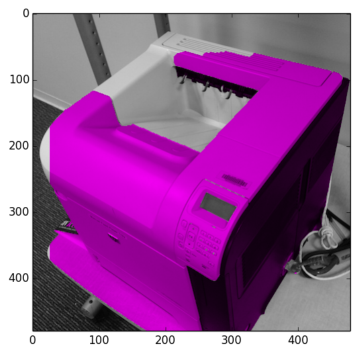


Figure 9: Final results showing segmented image for HP Printer.

5 Conclusions

Overall, we were very satisfied with the performance of our classifier. It often achieved 100% accuracy during testing, even on the bigger data sets, and when correct the histograms usually indicated a strong consensus towards the correct label. We would also like to discuss issues we see with the scalability of our algorithm to much larger data sets. Given the massive amount of time training took, we didn't run our algorithm on datasets with more then 20 different labels. However, even in the increase from 10 to 20 labels we noticed about a 10% drop in accuracy. After looking at the results and the images the classifier failed to recognize, a few common themes emerged for areas where it had trouble.

The first major issue was bad sets of training images extracted from the web. This was easy to control for, but for our algorithm to be able to recognize *any* object, it would need to scrape images corresponding to millions of product queries in an unsupervised manner and train itself on those images. This removes the long-term solution of manually combing through scraped image results. We weren't able to come up with a great solution to this, opting instead to take some pictures ourselves or manually remove bad matches from web data sets.

The next major area the classifier failed in was products with mostly uniform color. This pointed to a limitation of an algorithm based solely on SIFT descriptors; it relies primarily on differences in orientation and color gradients. A more robust algorithm would have to consider the shape of objects in an input image, and consider a sufficient amount of features to have different products cluster in distinct locations in the feature space. Our model was good at clustering for smaller data sets, but SIFT features didn't produce sufficiently distinct clusters as the number of labels in consideration increased.

References

[1] D. G. Lowe, *Distinctive image features from scale-invariant key-points.* IJCV, Nov. 2004.

[2] Fischler, Bolles, *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography.* Magazine Communications of the ACM, 1981.

[3] The OpenCV Library *Dr. Dobbs Journal of Software Tools* (2000) by G. Bradski

[4] Rother, et. al, *"GrabCut – interactive foreground extraction using iterated graph cuts*, (2004)