Hop Helper
A Beer Label Recognition Engine

Bryan Offutt
boffutt@stanford.edu
Stanford University
CS 231A: Introduction To Computer Vision

Abstract

When sitting down and enjoying a new beer many beer drinkers want to know everything they can about the beverage they have in front of them. Unfortunately, it often the case that this information is not readily or easily available on the bottle. With such a competitive craft brewing market, many breweries have begun to leave important information such as International Bitterness Units (IBUS), Alcohol By Volume (ABV), and even style off the bottle in order to make room for flashy, eye catching images. By utilizing SIFT descriptors, I present a simple and effective model for taking advantage of these flashy labels in order to provide a system that effectively identifies a beer from a user inputted photograph of the bottle. This identification of the beer name from a photo of the bottle creates a quick and easy way to provide users with information about the beverage before them.

1. Introduction

The past few years has seen a huge surge in the popularity of craft beers. A beer market once dominated by large corporations such as Aneuser-Busch and Miller Brewing Company is now segmented amongst thousands of smaller breweries, each of which brings dozens of beers to the table. As a result of this competition, many craft brewers have focused on making wild, flashy labels in order to make their beer stand out on a shelf of hundreds of competitors. Unfortunately for the consumer, this often means that important information about the beer can be difficult or impossible to find. For example, consider the label in Fig 1.

Figure 1.

This label is attractive and eye catching, with bright colors and a nice image in the center, but is not very informative. It tells us that the beer is an extra IPA, but lacks any information about the alcohol by volume or international bitterness units. Furthermore, it is not entirely obviously what the name of the beer is: Is it Sierra Nevada, or Torpedo? Or Sierra Nevada Torpedo? This lack of concrete information can be both frustrating and annoying to a beer drinker who wants to know everything they can about the beer they are about to enjoy. In addition, the ambiguity of the title of the beer can make it difficult (or impossible!) to remember the name of the beer should you want to purchase it again.

The motivation for this project was to solve these problems by utilizing the *advantages* that arise from flashy labels. While the desire to be eye-catching can often result in the labels being less informative, it also means that labels are likely to be highly differentiated. With so much effort going into being noticed, breweries are careful to make sure that their labels look different than any other beer on the market. By utilizing this differentiation, I propose a system that uses comparisons of SIFT descriptors in order to match a user inputted bottle to it's proper label. Once this label recognition is complete, we are able to return to the user not just the name of the beer, but also brewery information, IBUs, ABV, and more.

2. Previous Work

Label recognition has been applied in a

number of areas other than beer labels, the most closely related and popular example being Vivno. Vivino is a mobile application that allows a user to obtain information about a bottle of wine by taking a picture of the label. Vivino's results are pretty accurate, and it has gained a lot of popularity since it's release. Seeing this application made me wonder why there was not a beer equivalent, and created the motivation for this project.

3. Recognizing Beer Labels

The overall goal of my project was to create a system that, given a database of beer labels and a query image of a bottle, could identify which label the bottle most closely matched. Once a match was found, further information about the beer could be passed on to the user.

The first step in this process was to obtain the labels for the database. To do so, I used an open source database at www.brewerydb.com. This database contains over 15,000 beers in total, 10,114 of which had a corresponding label in the database. The photos in the database were of the labels peeled off of the bottle and laid flat, as can be seen in Figure 1. In addition to label images, this database also contained all of the information I needed to pass back to the user, including International Bitterness Units, alcohol by volume, style, brewery, and even ingredients. The photos of bottles used in testing were obtained both from crawling BeerAdvocate.com and from images taken by myself. In order to create as robust a system as possible, these photos were taken from varying angles with varying levels of illumination (Fig 2). In addition, some bottles were chosen that were of the same brand as a label in the database, but had a slightly different label.

Figure 2: Bottle Photos w/ Varying Angles and Illumination

After obtaining the images to be used in our database, SIFT (Scale-Invarient Feature Transform) [1] descriptors were calculated for each label, as well as for the query image (Figure 3). The fact that SIFT is invariant to scale and illumination made it an obvious choice for this project, since photos are likely to be taken in dimly lit bars and from a variety of distances away from the bottle. These SIFT descriptors were the very heart of my system, and were used in a variety of different methods in an attempt to find the best solution to the problem of label recognition.

Figure 4: A Beer Label With It's Sift Descriptors

3.1 Training → Test Feature Matching

My initial attempt at a recognition system was intended to give me an idea of what would and would not work on a high level, and thus was very simplistic. In this method the descriptors for each training label were matched to descriptors in the input image using a simple nearest neighbors (or shortest Euclidian distance) calculation. If the first nearest neighbor found using this method was greater than 4/5 the size of the second nearest

neighbor, then this match was thrown out. After running through this method for each descriptor in a training label, the total number of valid matches for that label was tallied. After completing this for every beer in the database, the beer with the highest number of valid matches was chosen as the best match for the input image.

While simple and fairly easy to implement, this method was not very effective. This lack of efficacy was due mostly to the fact that multiple key points in a training label would frequently map to the same point in the test image (Figure 5). Even worse, it seemed that the most common matches were between key points that were not relevant to the actual content of the label. This can be seen in Figures 4 and 5. Despite having nothing in common with the label on the bottle in the query image, these two training labels were selected as the two of the best matches because multiple key points in the labels matched to the edge of the bottle. Even though they tell us nothing about how close the labels are to one another, these matches are still considered valid, resulting in skewed results and poor accuracy.

Figure 4-5: Poor Matches Between Edges

3.2 Test → Training Feature Matching

In order to solve the issues caused by arbitrary and uninformative matches, a second method was implemented. This second method was similar to the initial attempt in that it involved matching between SIFT features, but was different in how it went about doing it. Rather than matching each individual training descriptor to a descriptor in the test image, it instead matched each descriptor in the *test* image to a descriptor belonging to one of the training labels.

The first step in this method was again to calculate the SIFT descriptors for each of the training labels. Once calculated, the descriptors for all labels were then merged together into a giant bank of all descriptors. In total, this bank contained more than 5 billion descriptors. Each SIFT descriptor in the test image was then matched to the descriptor in this bank whose Euclidean distance from the test descriptor was smallest. In this case I did not use the second nearest neighbor check as I did in the initial method because it is possible for two different beers from the same brewery to have very similar features (Figure 6). Since the distance between these two features is likely incredibly small, using the second nearest neighbor check would cause these two beers to trample out one another's valid matches. Once a match is found for each feature in the test image, the number of features matched in each training image is calculated and the image with the highest number of correspondences is chosen as the most likely beer match.

Figure 6: Two Beers From the Same Brewer with Similar Labels

This method was incredibly effective, but very slow. As stated above, the feature bank used in this method contains more than 5 billion descriptors. Each of these descriptors is a vector of 128 32-bit floats. As a result, storing this bank took over 2 gigabytes of memory. Unfortunately, since I was running this system on a machine with

just 1GB of memory, it was impossible for me to store the entire bank at one time. The only solution to this problem was to re-calculate the SIFT descriptors for each training label every time I wanted to compare them to a descriptor in the test image. Computing descriptors this many times took 19 minutes *without* doing any other computations (additional results can be found in supplementary material), making it clear that a more tractable method was necessary.

3.3 Test → Training Using a Subset of Features

My initial solution to this problem was simple: rather than use ALL of the descriptors from each training label in the bank, I instead randomly selected 20 descriptors from each label. These 20 descriptors for each label were then added to the bank, resulting in a bank of just 202,280 descriptors, significantly less than the 5 billion descriptors in our original method. Since these descriptors took up just a tenth of a gigabyte, the entire bank could be easily stored in memory. This significantly cut down on running time, since the bank could be calculated one time and then stored in memory for subsequent runs. The smaller bank further cut down run time because each test descriptor had to be compared to just over 200,000 descriptors rather than a whole 5 billion (approximately 25,000 times less). Overall this method presented a faster, more space efficient solution to the problem while maintaining relatively acceptable level of accuracy (more information can be found in supplementary materials).

3.4 Test → Training Using K-Means Clustering and Bag Of Words

While creating a bank from a subset of label descriptors significantly improved the tractability of my system, I wondered if I could make it even faster. The improved system still took a few seconds to run, an amount of time that would probably be considered too long in a commercial application. In light of this, I wanted to find a way to cut down running time to less than a second or two.

In order to do this, I utilized K-means clustering and a bag of words model. By taking the subset of label descriptors discussed in the 3.3 and running k-means clustering (I would have liked to do this using the bank containing ALL of the features, but was again limited by tractability issues), I was able to group all of the descriptors into just 1000 clusters. The center of these clusters became my bag of words. I then used the same nearest neighbors matching discussed in 3.2 to match each training label descriptor with it's nearest neighbor in the bag of words. How many times each centroid in the bag of words was matched for a particular label was then tallied in order to create a histogram for each training label. Each histogram was then normalized in order to ensure that the sum of it's entries was one.

When given a training image, the same steps were executed in order to create a training histogram. This test histogram was then compared to the histogram for each training label and the training label with the histogram most similar to that of the test image was selected as the best match. This method significantly reduced the run time of my system, but ended up really crippling the accuracy (more information on results can be found in the next section).

4 Results

In order to test the accuracy of the proposed methods, I ran a set of 50 bottle images through systems 3.2, 3.3, and 3.4. This test set included photos taken at a variety of scales and illuminations. In order to further ensure robustness, bottles whose labels shared features with other beers in the database (Figure 6) were included, as were cans and bottles whose labels slightly differed from the corresponding database label. Because Beer Advocate often had incorrect bottle images, the test images had to be hand selected to ensure that they matched a label in the database.

Each system was trained using the dataset from brewerydb.com as discussed in section 3. All methods were trained using all 10,114 beers with the exception of the method discussed in 3.2.

Correct Guess Result	Lone Winner	Tied For First	Second	Total % Accuracy
Full Features for Subset of 512 (3.2)	48	0	1	96%
20 Features for All Beers (3.3)	22	10	3	64%
K-Means Clustering (3.4)	2	0	0	4%

Figure 7: Results For Three Methods

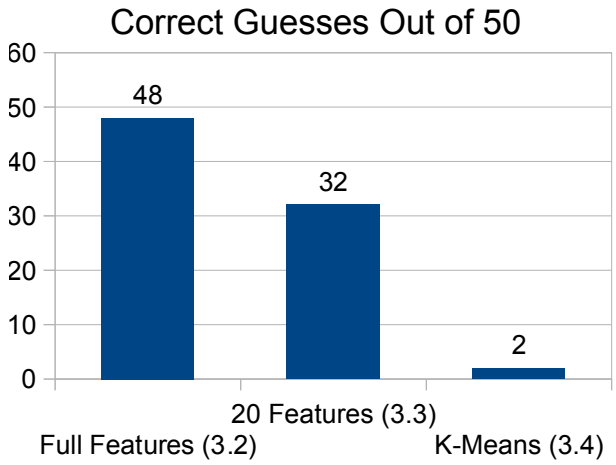

Figure 8: Correct Guesses Out of 50

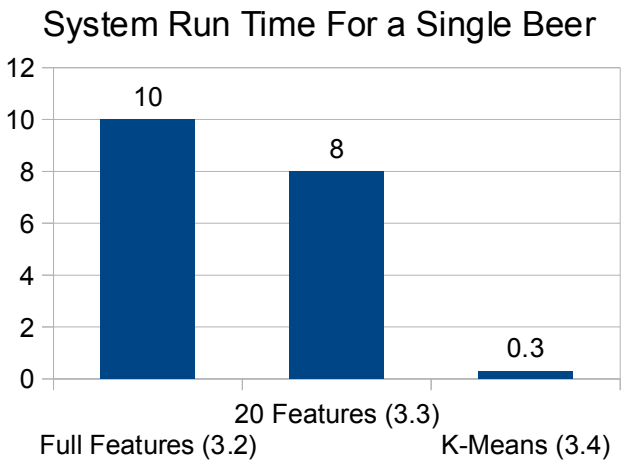

Figure 9: Run Times for Single Beer

Because this method took over 20 minutes to run per beer when using the full data set, this system was instead trained using just the first 531 beers in the database.

As can be seen in figures 7,8, and 9, each of the three methods had very different results. The method proposed in 3.2 was the most effective, correctly identifying 96% of the beers in our test set. Of the two tests where the correct beer was not chosen, one had the correct choice as the runner up. Also worth noting is that both the runner-up and incorrect primary guesses in this case had the *exact* same label with the exception of the text identifying the beer name. For the second incorrect test example the system predicted the correct label as being the fourth most likely choice. All three of the beers in front of the correct label were from the same brewery, and thus had very similar bottles.

While this method was incredibly accurate, it also ran the slowest of the three, taking approximately 10 seconds to run the system. This number is even worse when you consider the fact that this method was run on a subset of just 531 beers rather than the entire 10,000. Thus, if it were able store and run on the entire set the run time would be roughly 20 times longer, making it the slowest by over three minutes.

The method proposed in section 3.3 was the most balanced of the three approaches. Despite only using 20 features from each training label, this method was able to correctly identify the beer in the test image 64% of the time. 44% of the time the system chose the correct beer as the *only* number one choice, while remaining 20% of the time the correct beer was tied for the most likely spot. These “ties” for first place were usually shared between 2 to 4 beers. In a real life application these ties would be nearly as good as singling out a best choice, since you could return to the user the images all of the beers tied for first and allow them to select which beer is theirs. This method also ran relatively efficiently, and was able to make a prediction after approximately 8 seconds.

Unlike the first two methods, the k-means method discussed in section 3.4 was not very accurate at all. When run on our test set of 50 beers, this method was only able to correctly

identify two of them, while the remaining 48 didn't even have the best guess in the top 5. Though these results may seem shocking at first, they actually make pretty good sense. Boiling over 5 billion features down into just 1000 clusters gets rid of a lot of information, and can make different labels resemble one another much more than they actually do in real life. By increasing the number of clusters to a larger number, more accurate results could be obtained (unfortunately, clustering into 1000 clusters already took my computer 10+ hours!). However, this method was effective in speeding up our system, and was able to make a prediction in just . 3 seconds. Unfortunately, this speed is not nearly enough to make up for such horrible accuracy.

5 Conclusions

Overall, I think the results of this project are promising. Though it may not have been blazing fast, I was able to achieve my goal of developing a system that can identify beer labels with incredible (90%+) accuracy. This method was limited only by the hardware it was running on, and could be executed significantly faster given a machine with more memory and computing power, something that is relatively easy to come by with modern day cloud infrastructure. Given a machine with a lot of memory and a quick processor this system could go from taking 20-30 minutes to being executed in just a few seconds.

In addition to this, I was also able to develop a faster, lighter weight system that was able to maintain greater than 60% accuracy despite running in under 10 seconds. This result is especially promising when you consider the fact that the system is choosing these correct labels out of a pool of over 10,000. All in all, I believe that these two systems provide accurate, tractable solutions to the problem of beer label recognition.

6. References

[1] Lowe, David G. (1999) 'Object Recognition from Local Scale-Invarient Features". Proceedings of the International Conference of Computer Vision 2. pp. 1150-1157. doi:10.1109/ICCV.1999.790410

[2] The OpenCV Library. Dr. Dobb's Journal Of Software Tools (2000) By G. Bradski

[3] www.BreweryDB.com

[4] www.Beeradvocate.com