B657: Computer Vision
Assignment 3: Object Detection
Spring 2016

Charlene Tay (ctay)
Debasis Dwivedy (ddwivedy)
Tousif Ahmed (touahmed)

March 28, 2016

1. Introduction: Food Image Classification

Food is a very important part of society and culture, and understanding food habits has been considered in many areas of research, especially in consideration of the health impacts of diet and food habits. One particular concern in the United States is the growing rate of obesity in American adults. In order to address this crisis, people are increasingly looking for automated ways to measure dietary and supplement intake in obesity study and treatment. One potential solution to this is in the use of computer vision to recognize food images (via mobile/wearable cameras) and associating it with properties such as nutritional information, which can potentially help with assessing and tracking dietary habits.

The goal of this assignment is to apply various recognition algorithms to food images and to assess their effectiveness in food classification. This is an interesting challenge as food presents high variability in appearance. We were provided with a data set of training and test images for 25 different food categories. In the following report, we outline the methods of the various algorithms implemented, and compare the performances between methods.

(a) Example of an image from the data set provided

2. Part 1: A Simple Baseline

How the code works

```
./a3 mode baseline
```

where `mode` is either `train` or `test`.

Implementation Details

In order to establish a baseline for the comparison of the multiple methods that we implemented, we first set up a support vector machine (SVM). This program subsamples the images to a fixed size, 40 by 40 pixels, then converts each image into a vector by concatenating all the rows of the image. An SVM library is then called to train the SVM on this task. For this assignment, we chose to use the recommended library, SVM_multiclass (see https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html).

Our program writes out a file in the format expected by the SVM library, makes a **system()** to execute the program on the command line, and then parses the output files produced by svm_multiclass. Given a new image, the testing program applies the SVM to estimate the correct class for each test image.

We have included in the repository our svm multiclass directory. We made ensure that before making the system call, it creates the binaries.

Analysis

The Support Vector Machine library (svm_multiclass) was implemented successfully. By tweaking the value (c) for the trade-off between the training error and margin, we were able to get better values for the baseline SVM. To further try to improve it, we also experimented with tweaking the image size of the input images, as well as using color images instead of gray-scale. We found that the best image size is 40 x 40 pixels - resizing to 20 (precision = 10%), 80 (precision = 12%), or 100 (precision = 13%) did not yield better results. Using colors as additional dimensions for information from the images also did not improve the results.

Table 1: Effect of different c-values on svm baseline

c	**Precision**
30	10%
40	11%
50	13%
75	12%
100	11%

3. Part 2: Traditional Features

As the goal of this assignment is to understand different features. We did not make any changes to the SVM parameters.

3.1 Eigenfoods

Command

```
./a3 mode eigen
```

where `mode` is either `train` or `test`. The model file is stored as eigen_model.

Implementation Details

The tasks represents the way in which we can recognize, verify, and identify features. In this case, it is food items. The bigger problem of identifying features is that even two same food items might appear completely different under different lighting conditions. This makes it difficult to differentiate between two distinct items as the the pixel intensities of the image may be similar. An image with high dimensions can be very computationally intensive and if real life images are considered, it could be said that it can take an exponential amount of time. To address this problem, we apply principal component analysis (PCA) techniques, where the dimensions of the image are reduced. We can represent the original image using projection techniques. This projection shows us the difference between the original image and the dimensionally-reduced image.
Steps followed during implementation:-
1. Convert the RGB image to gray scale image.
2. Convert the original image (MxM) to a lower dimension image (NxN), say for example if the image if of 382x382 dimension, resize it to 40x40.
3. Normalize and flatten each image to a single dimension vector of length size M^2.
4. Form a square matrix (MAT) of all the test images. For example, given 50 images, the matrix will be of 50 x M^2 size. Each row represent a vector image.
5. Build a co-variance matrix from the vectored image MAT x MAT^T.
6. Decompose the matrix into eigen vector and eigen values. I have used Jacobi decomposition algorithm for this purpose.
7. After calculating the eigen vectors and eigen values we normalize it and project it to image space.
8. We store the projection as a model for our classifier to classify the images.
9. The model is stored as eigen_model. This is passed to the SVM system along with the feature vector of the test image to classify the image.
10. The output is stored in a text file.

Problems Faced

While reducing the dimension of the image the accuracy of the image was extremely low compared to bag of words method. A higher accuracy could be reached if we increase the feature space but then it becomes computationally intensive.

Analysis

The eigen decomposition reduces the dimension of an image to lower dimension and treats it as feature vectors. Since the dimension is reduced the computation speed increases but at the cost of accuracy. I could reach a maximum accuracy of 8.7% .As we increase the dimension of the image the computation speed increases many fold. So overall I did not find this method suitable for object recognition problem.

Table 2: Testing various k-values,feature space and percentage accuracy

k	Reduced Image	Precision
20	40x40	4%
40	20X20	4.4%
60	50x50	5.2%
80	40x40	8.7%
120	40x40	6.4%

3.2 Haar-like features

Command

```
./a3 mode haar
```

where `mode` is either `train` or `test`. The model file is stored as haar_svm_model.

Implementation Details

Implementing haar was challenging. Although, we used similar functions, still we were getting segmentation faults because of the length of the feature values. To solve this problem, we normalized the features.

We needed to make several design decisions for this part. We resized the images and made it smaller. Also we needed to tweak the size of the base detectors. There were several parameters in this process, which made it extremely difficult to work. As the training was slow, so it is quite difficult to test all the parameters.

Overall, we believe that Haar-like feature is not good feature for this problem. As foods can be of different shapes. Morover, the size of the base detector was completely random, which can be the reason of poor accuracy of the bag of words.

We mainly used grayscale images for this part at the beginning. When we added color, the accurancy improved significatly. We also manipulated the number of features. Lower number of features gives poorer results.

Problems Faced

We faced a memory problem. We could not find any solution for that. When we have 1000 features, if we make the image size more than 100 we experienced segmentation fault. When we did not resize the image, we needed to lower the number of features. Resizing the image to 40 × 40 solved that issue.

Analysis

While we tried grayscale, Number of Features 1000 gave an Accuracy of 3.6% which was below baseline. This result motivated us to try color and changing the number of features.

Table 3: Quantitative Analysis on Haar Like Features

No of Feature	**Precision**
100	8.4%
500	4.6%
1000	3.4%

Overall the accuracy was really poor. Table 3 shows some result.

We think the accuracy for haar based feature is completely dependent on the generated feature.

3.3 Bag of Words

Command

```
./a3 mode bow
```

where `mode` is either `train` or `test`. The trained svm model file is stored as "bow_svm_model".

Implementation Details

The Bag of Words model is widely used in natural language processing techniques for text documents. In document classification, a 'bag of words' is a vector of occurrence counts of words that appear in the document. When applied to image classification, the image is represented as a document, and image features are treated as words. The model can thus take the form of a histogram representation of the image, based on a collection of its local features. For the purposes of classification, the images can then be compared and categorized based on this discrete and compact histogram representation.

Four key steps were taken to train the Bag of Words model for image classification. First, local features were extracted from the set of training images. Scale Invariant Feature Transform (SIFT) descriptors were extracted from each image and written into a large file, where each row of the file represents one descriptor. Each column in the row gives the training image that the descriptor is from, the class/category of the training image, followed by each value of the 128-dimension SIFT descriptor. Next, a K-Means algorithm was used to cluster the raw SIFT descriptors into k visual words. Here we used an existing C++ implementation of k-means clustering called "Yakmo" (developed by Naoki Yoshinaga, see http://www.tkl.iis.u-tokyo.ac.jp/ ynaga/yakmo/). A binary file for Yakmo has been included in the repository for use on the SOIC linux machines (Tank/Burrow). The program takes in the data file containing all the SIFT descriptors, clusters them, and outputs two files - one with the trained cluster centroids ("kmeans_centroids.txt"), and one file with the clusters that each of the SIFT descriptors have been assigned to ("kmeans_output.txt").

Each cluster centroid represents a visual word in the vocabulary of k distinct features obtained from the training data. The clustering algorithm maps each descriptor to a visual word in our vocabulary, and we counted the frequency of each mapping in each image, thus obtaining a k-dimensional vector that is a histogram representation of that image. These vector representations are stored in a file named "bow_histograms.dat". Finally, we take these histogram representations of the training images and used that as inputs for training our Support Vector Machine model for classification. The trained SVM model is saved as the file "bow_svm_model".

When testing the model on a new set of images, the test images are put through a similar process as in the training stage. The images are read in, converted into gray-scale and a set of SIFT descriptors are extracted from each image. Using the set of cluster centroids ("kmeans_centroids.txt") previously obtained from the training images, the test image SIFT descriptors are each mapped to a cluster. As with the training images, the clustered points of the test images are then represented as histograms and put through the SVM for classification.

Analysis

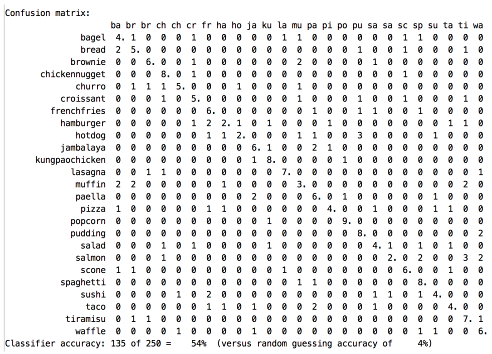
Using the Bag of Words model for representing images works very well with the Support Vector Machine model. Experimenting with various k values, we found that the optimal value for this purpose is around 500. With this value, we were able to correctly classify 54% of the set of 250 test images. Table 3 shows the results obtained with the various k values tested. However, we found that training the model was taking an incredibly long time when using the original images. We decided to experiment with rescaling the images down to a smaller size and train using these smaller images. However, this method yielded poorer results than using the original image sizes so we stuck with our original method.

Table 4: Testing various k-values for Bag of Words, full image size

k	**Precision**
100	45%
250	51%
500	54%
750	54%
1000	50%

3.4 Comparing Methods

Overall, we had the most success by using the Bag of Words model for food image classification. This method successfully classified 112 out of 250 test images, yielding a precision of 54%. This is in comparison to our results from implementing Principal Component Analysis (PCA/Eigenfoods), which yielded a precision of 9%, and the Haar-like features implementation, which yielded a precision of 8.5%. There are a few hypotheses as to why this is so. We think that Principal Component Analysis (PCA) involves a



```
Confusion matrix:
               ba br br ch ch cr fr ha ho ja ku la mu pa pi po pu sa sa sc sp su ta ti wa
         bagel 4. 1  0  0  0  1  0  0  0  0  0  1  1  0  0  0  0  0  0  1  1  0  0  0  0
         bread 2  5. 0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  0  0  0  1  0
       brownie 0  0  6. 0  0  1  0  0  0  0  0  0  2  0  0  0  0  1  0  0  0  0  0  0  0
  chickennugget 0  0  0  8. 0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0
        churro 0  1  1  1  5. 0  0  0  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0
     croissant 0  0  0  1  0  5. 0  0  0  0  0  0  1  0  0  0  1  0  0  1  0  0  0  1  0
   frenchfries 0  0  0  0  0  0  6. 0  0  0  0  0  0  1  0  0  1  1  0  0  1  0  0  0  0
     hamburger 0  0  0  0  0  1  2  2. 1  0  1  0  0  0  1  0  0  0  0  0  0  0  1  1  0
        hotdog 0  0  0  0  0  0  1  1  2. 0  0  0  1  1  0  0  3  0  0  0  0  1  0  0  0
     jambalaya 0  0  0  0  0  0  0  0  0  6. 1  0  0  2  1  0  0  0  0  0  0  0  0  0  0
kungpaochicken 0  0  0  0  0  0  0  0  0  1  8. 0  0  0  0  1  0  0  0  0  0  0  0  0  0
       lasagna 0  0  1  1  0  0  0  0  0  0  0  7. 0  0  0  0  0  0  0  0  0  0  0  0  1
        muffin 2  2  0  0  0  0  0  1  0  0  0  0  3. 0  0  0  0  0  0  0  0  0  0  2  0
        paella 0  0  0  0  0  0  0  0  0  2  0  0  0  6. 0  1  0  0  0  0  0  1  0  0  0
         pizza 1  0  0  0  0  0  1  1  0  0  0  0  0  0  4. 0  0  1  0  0  0  1  1  0  0
       popcorn 0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  9. 0  0  0  0  0  0  0  0  0
       pudding 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8. 0  0  0  0  0  0  0  2
         salad 0  0  0  1  0  1  0  0  0  0  1  0  0  0  0  0  0  4. 1  0  1  0  1  0  0
        salmon 0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  2. 0  2  0  0  3  2
         scone 1  1  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  6. 0  0  1  0  0
     spaghetti 0  0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0  0  0  8. 0  0  0  0
         sushi 0  0  0  0  1  0  2  0  0  0  0  0  0  0  0  0  0  1  1  0  1  4. 0  0  0
          taco 0  0  0  0  0  0  1  1  0  1  0  0  0  2  0  0  0  1  0  0  0  0  4. 0  0
      tiramisu 0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  7. 1
        waffle 0  0  0  0  1  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  1  1  0  0  6.
Classifier accuracy: 135 of 250 =    54%  (versus random guessing accuracy of     4%)
```

(a) Bag of Words: Confusion Matrix Output. 54% precision with k=500

reduction in dimensionality of the images in the feature space. In addition, Haar-like features might work poorly with foods because the features of foods do not necessarily fit into rectangles and will not work with rectangle filters. The Bag of Words model works well because it creates representations of each image by using the distinctive features of each image.

4. Part 3: Deep Features

Command

```
./a3 mode deep
```

where `mode` is either `train` or `test`.

We also included our overfeature directory and when its called it is downloading the weights. The Model file is stored as deep_svm_model.

Implementation Details

It was the most time consuming part as it took time to create the training. Generally, it crates huge amounts of features. Therefore, we resized the images to keep the number of features in control.

Otherwise, the implementation was straight forward.

Analysis

We were able to run the full training couple of times. We resized the image to 270 $\times$ 270 and achieved an accuracy of 8.4%.