CSCI-B657: Computer Vision
HW3: Food Detection

Yisu Peng (yisupeng@indiana.edu)

Marshal Patel (marshalp@indiana.edu)

Ishtiak Zaman (izaman@indiana.edu)




Introduction

In this assignment, we implemented an application that takes food images as input and recognizes the food. We explored different types of feature extraction of objects and trained an SVM classifier to recognize the food item. We used the following features:

- Baseline Feature (Part 1)
- Eigen Foods (Part 2.1)
- Haar like Feature (Part 2.2)
- Bag of Words (Part 2.3)
- Deep Feature (Part 3)

Baseline Feature (Part 1)

For quantitatively, this method works pretty fast, since it's quite simple. On the other hand, the quality is acceptable. For the gray case, it's two times to the random method. For the color case, it's better.

Accuracy: 14% color, 9.6% gray

Remarks of accuracy:

The method using color information has better performance than gray scale method. I think the reason is that it's using more information/features than the gray scale case.

Compile: make

Train: ./a3 train baseline

(Training may take around 12 minutes, we already provided the trained files in the

repository, this step can be skipped).

Classify: ./a3 test baseline

Eigen Foods (Part 2.1)

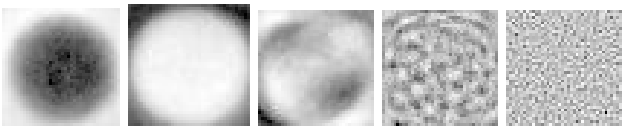
For this part of the assignment, training images were pre-processed to grey-scale images and resized to equal size (40 x 40). Each pixel was treated as a feature. Hence we had a total of 1600 features for each image. Hence, the size of the total dataset is 1250 x 1600.

In order to reduce dimensions of the dataset, Principal Component Analysis was performed using the Eigenvalue decomposition. Each eigenvector has a dimension of 1600 x 1. Top k (900, 1000, 1250) eigenvectors were selected and SVM model was trained with different values of c (1,0.1, 0.01).

For test set similar process was followed to extract features and classify using the model built using training.

The eigenvalues decreased abruptly for k>1250.

Few of the top k vectors look like following. These images are present in **rolled_images** directory:



Accuracy:

	900	**1000**	**1250**
1	**6.4%**	**5.6%**	**4%**
0.1	**6%**	**6.4%**	**7.2%**
0.01	**8%**	**8%**	**8%**

Remarks of accuracy: Highest accuracy was achieved with k = 1250 and c = 0.01.

Compile: make

Train: ./a3 train eigen

(Training may take around 5-7 minutes, we have already provided the trained files in the repository, this step can be skipped).

Classify: ./a3 test eigen

Haar like Feature (Part 2.2)

In the haar-like feature extraction we have 10 different types of filters. The filters are: 1x2, 1x3, 2x1, 3x1, 2x2. Each filter has black and white regions. We flipped the colors to make it total 10 types of filters.

During the training, we randomly generated 1000 features with random type, size and location; and stored them in a file so that we can reuse the filters again during classification. For each images of the training, we resized the image to 60x60, find integral image values for all 3 colors using dynamic programming, and get 3000 feature values on that image. For each color of the image, we have 1000 feature values, computed by using the 1000 haar like features. In total, every image has 3000 feature values. The feature values are too diverse at this point, so we normalized the feature values within the range [-100, +100]. At this point, we trained an SVM for all the images with those 3000 normalized feature values with the known label of that image; and stored the learned SVM model file in the disk.

During the classification, we again resized the image to 60x60. Read the feature values from the disk, computed integral image for all 3 colors using dynamic programming, and generated the 3000 feature values of the image and normalized them. Then we classified the image by calling the SVM classifier binary with the learned model files and returned the class label to the calling process.

Accuracy: 26% with colored image, 16% with gray-scale image.

Comparison with Part 1: Part 1 works with pixel by pixel, which is not a very good method for classifying. The Haar like feature based classification worked much better than that due to region based feature computation.

Compile: make

Train: ./a3 train haar

(Training may take around 5-6 minutes, we already provided the trained files in the repository, this step can be skipped).

Classify: ./a3 test haar

Bag of Words (Part 2.3)

For the bag of words algorithm, first we need to do the SIFT on each picture and get all the SIFT descriptors. After that, in order to find the words for a dataset(all pictures from different classes), we need to do a clustering method to find the centers among all those feature points. Here for the given training dataset, we have about 3 million feature points, that's a huge number. The clustering algorithm we using is simply the K-means algorithm, and the K value we choose is 20 based on our experiments. We tried 5, 10, 20 centers and find 20 have best performance among these values. We think that 20 centers are enough, more centers may cause overfitting problems in later SVM. We are using existing K-means algorithm in the OpenCV library.

Then we use these 20 centers to label all those 3 million features, by finding the most closest center for each feature on Euclidean distance. Then, for each image, we count the number of features that belongs to each center and we get 20 numbers, and we do a normalization on these counts. After this step, we get a 20-d vector for each image in the dataset. Use these 20-d vectors we train the svm.

Accuracy: 16%

Comparison with Part 1:

This method is better than both the color case and gray scale case in the Part 1.

Compile: make

Train: ./a3 train bow

(Training may take around 40 minutes, we already provided the trained files in the

repository, this step can be skipped).

Classify: ./a3 test bow

Comparison between Haar like feature vs Bag of Words vs Baseline feature:

The Haar method has 26% accuracy, the bag of words method has 16% accuracy, while the baseline feature has 14% of accuracy. Haar detection worked much better than bag of words method we implemented on this dataset.

For haar like feature, the accuracy of 26% is quite good. Haar like feature works better to recognize human faces. And different foods have same kind of regions, in which cases the haar like feature will identify as similar. And in the given training dataset, some images have only one piece of the food, and some images have group of the foods. In this situation, the haar like feature will not classify very well. Colored image produces a much better accuracy than gray-scale image, because different food has different colors.

For bag of words, the accuracy of 16% is not very good compared to Haar like feature. One of the reasons is that the SIFT features of this dataset are distributed averagely across the space, which don't have obvious clustering centers or boundaries. This causes a problem for the k-means clustering method to find reasonable centers. If the centers are not good, then the histogram might not make much sense to classify the images.

For baseline feature, the method using color information has better performance than gray scale method. The reason is that it's using more information/features than the gray-scale case. Also it works with pixel by pixel, which is not a very good method for classifying foods.

Deep Feature (Part 3)

We used Overfeat library for feature extraction. We resized the training images to 231x231. The overfeat library requires the images to be at least 231x231. The overfeat library already has ImageNet trained network. The network has 22 layers. We extracted the feature from layer 21, the layer before the output layer.

For training, we resized the images to 231x231, and used the overfeat binary to extract the features from the 21st layer output. At the 21st layer output, there were 1000

features each with h=1, w=1, thus holding only one cell per feature. With these features we trained an SVM.

For the classification, we again resized the images to 231x231 and extracted the 21st layer output from the overfeat binary. Then used the trained SVM model to classify the images with the extracted deep features.

Accuracy: 70%

Comparison with Part 1 and Part 2: Much better accuracy of 70% compared to all other methods we used. The overfeat network is heavily trained, and we used a very deep layer that can differentiate between different items and extracted better quality features which results in a better classifier. If we would use the last layer of the network, we might get even better result.

The training and testing was not as fast as some of the methods from Part 1 and 2, but the final classification quality is much better than other methods.

Compile: make

***OVERFEAT Library Required:** To run the program we need overfeat library in the directory. The default location of the overfeat binary file is *"/overfeat/bin/linux_64/overfeat"*. This location can be changed from the 1st line of *Deep.h* file.

#define OVERFEAT_PATH "/overfeat/bin/linux_64/overfeat"

Train: ./a3 train deep

(Training may take around 50 minutes, we already provided the trained files in the repository, this step can be skipped).

Classify: ./a3 test deep

Conclusion

In conclusion, we tested our food detection application with different feature extraction methods and compared their approaches and advantages/disadvantages. Some of the methods are not very ideal for food feature extraction and did not get very good accuracy, while other methods such as using convolutional neural network for

feature extraction and training an SVM worked pretty well.

References

1. SVM Multiclass: https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html
2. OverFeat: cilvr.nyu.edu/doku.php?id=software:overfeat:start
3. SIFT++ Library
4. Lecture Notes and Slides
5. Wikipedia
6. OpenCV