B657 Computer Vision

Object detection

Assignment 3 -Report

Manikandan Murugesan

Rakibul Hasan

Prakash Rajagopal

Spring 2016

Simple Baseline

Part 1.1 - Basic SVM

We experimented with the Nearest Neighbor Classifier given and then we tried to implement the SVM. We used SVM_Light Multiclass provided by Cornell. (https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html)

We created a class SVM that inherited "Classifier" similar to "NearestNeighbor". We subsampled the images to a fixed size of 40x40 and converted into a 1600-dimensional vector. We used a system() call to invoke the SVM. The SVM Library classifies and gives us the prediction output.

Then we ran a Python program which computed the score and displayed a confusion matrix.

To run this part, the user has to give the following input:

To train : **./a3 train baseline**
To test : **./a3 test baseline**

We got a accuracy of **18.4%** for the test data with the training data provided.

The program took around **20 minutes** to **train** and **2 mins** to **classify**.

The above output is what we got when we considered color images.

When we converted the images to grayscale, there was a significant drop in the accuracy even though the running time was less.

We got around **10%** accuracy for grayscale images. The comparison in the output can be found in the file "SVM.txt"

Difficulties faced:
In this part, we had to figure the format in which we have to write the feature vectors to the file. For this, we had to check the example files as well as read the documentation provided by the SVM library.

Traditional Features

Part 2.1- Eigen Vectors

In eigen faces training part:

- Sample all the training images and create a vector for each training image
- All the vectors are concatenated to form a resultant matrix with with M columns of size n x n , where M is number of images and n is the height and width of re sized image
- Get average of each row in the matrix and subtract it to get matrix A
- We need to get the co-variance matrix , but instead of directly multiplying A and its transpose , the following optimization is done for efficiency.
- We multiply the transpose of the matrix with the matrix to get a resultant matrix
- We get the eigen vector and eigen values of this matrix
- We get the inverse square root of the eigen values and divide the projection of the eigen vector into A by this matrix to get the required eigen vector
- Finally we normalize it by its mean and convert it into required form for SVM training.

To run this,

```
./a3 train eigen
```

Classification:

- Project each image to to eigen vector space and then convert it into SVM format

- Finally the SVM provides the closest match which is returned

To run this,

./a3 test eigen

Part 2.2 - HAAR

How to run:

Train: **./a3 train haar**

Test: **./a3 test haar**

Note that since we are creating a single feature file that contains features for all classes, during training, we store the class labels in a file called “class- names”. We have uploaded this file in github along with the model file generated by svm. If the program is executed in test mode without first training, then please make sure the file containing class labels is copied to the working directory. If training is performed first, then this file is generated again.

Discussion:

Instead of generating all filters randomly, we have generated some filters that captures the shape of the food objects, along with random filters. We have mean normalized the input images, and performed unit vector normalization of the feature vectors. Also we have used colors for classification. However, using the filters it was really difficult to get good accuracy (best we got is 13%). One thing we noticed from the generated confusing matrix is that, for some reason, svm classifies spaghetti and scone a lot even if the actual classes are different.

Part 2.3 - Bag of Words

For Bag of Words, to improve performance, we blurred the image and resized the image to 1/8th of it’s size. We used the given SIFT library to detect the features and found several patches in the images. Then we found the codebook by performing k-means clustering over a cluster size of 1500 and ran it over 100 iterations for it to converge. We built the histogram and saved it to the training file.

For testing, we extracted the descriptors and passed it to the trained SVM classifier which predicted the type of food.

Then we ran a Python program which computed the score and displayed a confusion matrix.

To run this part, the user has to give the following input:

To train	: **./a3 train bow**
To test	: **./a3 test bow**

We stored the visual codebook to the file and uploaded it so that the user need not wait for the program to run the whole process.
To train from pre computed data : **./a3 train bow false**

The command in turn invokes the following system calls:
./svm_multiclass_classify bow-test-features-upload bow-model-upload bow-predictions-upload
python score.py bow-predictions-upload bow-test-features-upload

To test from pre computed data : **./a3 test bow false**

./svm_multiclass_classify bow-test-features-upload bow-model-upload bow-predictions-upload
python score.py bow-predictions-upload bow-test-features-upload

The accuracy we got for this method is **20%** when we kept the cluster size to 1500**.**

However the running time for the training took around 30 minutes to complete.

Difficulties faced:
We experimented with all the parameters. When the cluster size was less and the number of iterations for k means were less, the program took significantly less time

to run, but the accuracy took a hit. So we trained the data and attached the learned data so that the user can do an offline run.

Deep features

Part 3.1- CNN

Before running :

```
make init
```

Run

```
./a3 train deep
./a3 test deep
```

Description :

We use the feature detection of the overfeat package and feed it into the SVM to get a feature string. We basically resize the image to 231 x 231 which is the minimum size for the smaller network and get a vector of features per training image. We then use SVM learning on this set of vectors. For classification, we compare the feature set of the image with the model using svm_classify to get a final result. We got around 10% accuracy for CNN though we could have improved it with some extra training.

REFERENCES:

1 - http://cilvr.nyu.edu/doku.php?id=software:overfeat:start).
2 - https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html
3 - https://www.youtube.com/watch?v=iGZpJZhqEME