Assignment 3

CSCI B657 – Computer Vision

ssurendr-cshelke-dipiband

Shwethambari Surendran
Cyril Shelke
Dipika Bandekar

1. **FILES SUBMITTED:**

SR. NO	FILE NAME	INCLUDED/MODIFIED
1	a3.cpp	MODIFIED
2	Classifier.h	MODIFIED
2	Makefile	MODIFIED
3	svm.h	INCLUDED
4	deep.h	INCLUDED
5	haar.h	INCLUDED
6	pca.h	INCLUDED
7	bow.h	INCLUDED

- Classifier.h – Contains the code and core logic for Part 1, Part 2 Section 2, Part 3 Questions of the assignment
- Pca.h – Contains the code and core logic for Part 2 Section 2
- Bow.h- Contains the code and core logic for Part 2 Section 3
- Deep.h- Contains the code and core logic for Part 3

2. **HOW TO RUN OUR CODE?**

1. **PART 1:**

- **SECTION 3**

1. **Training SVM :** ./a3 train svm
2. **Testing SVM:** ./a3 test svm

2. **PART 2:**

- **SECTION 1**

    1. **Training SVM :** ./a3 train eigen
    2. **Testing SVM:** ./a3 test eigen

- **SECTION 2**

    1. **Training SVM :** ./a3 train haar
    2. **Testing SVM:** ./a3 test haar

- **SECTION 3**

    1. **Training SVM :** ./a3 train bow
    2. **Testing SVM:** ./a3 test bow

3. **PART 3:**

    1. **Training SVM :** ./a3 train deep
    2. **Testing SVM:** ./a3 test deep

Note: In general respective train, model, test and prediction files are generated along with the accuracies.

3. **ANALYSIS:**

SVM Learn and Classify

Function Used: (Section 1, 2 and 3)

void svm(const Dataset & filenames, string value)

Following is the brief outline of the algorithm

SVM train:

- Step 0: Each Input image in the train folder is taken as an input image
- Step 1: Each input image is resized (values used for testing 40, 50, 55, 60, 70, 30)
- Step 2: resized image is unrolled along x axis and vector of similar class is prepared
- Step 3: SVM multiclass learn package is used to learn images using system()
  References: https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html.

- Format of the input train file used : target(1-25) feature : value (pairs) #info

SVM test:

- Step 0: Each Input image in the test folder is taken as an testing image
- Step 1: Each input image is resized (values used for testing 40, 50, 55, 60, 70, 30)
- Step 2: resized image is unrolled along x axis and vector of similar class is prepared
- Step 3: SVM multiclass classify package is used to classify using system()
  References: https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html.

- Format of the input train file used : target(1-25) feature : value (pairs) #info
- Final resize value chosen = 50 * 50 (Accuracy 20% Obtained)

Answers to specific questions Part 1:

1. When images are resized using different dimensions various accuracies are generated
2. Images resized with color perform better as compared to the ones without color
3. As the resize dimensions increase above 60 the accuracy begins to decrease
4. For grayscale images accuracy is constant around 10.4% as shown in the table below
5. Model_file_svm has been added for reference

Find below the table with detailed analysis of different factors:

Sr. No.	Resize Dimensions	Accuracy%	Incorrectness %	Color	Time in sec	CORRECT
1	40x40	17.6	82.40%	YES	303	44
2	40x40	10.4	89.60%	NO	225	26
3	50x50	10.4	89.60%	NO	497	26
4	50x50	20	80%	YES	486.2	50
5	60x60	11.6	88.40%	NO	646.71	29
6	60x60	20.8	79.20%	YES	684	52
7	70x70	18.4	81.60%	NO	1227.25	46
8	55x55	19.6	80.40%	YES	543.38	49

1. EIGEN/HAAR/BOW

Section 1:

Randomly chose k to be 400 so that we had reduced features by 25%.

We used the symmetric_eigen() function available in CImg library.

When the eigen value were printed, we were able to see the following pattern:

The eigen values decrease from 9.4363e+08 to -1.5435e-05.

Function Used:

void train(const Dataset & filenames)

Following is the brief outline of the algorithm

For each image while training, we first converted into greyscale and unrolled it into 1 X 1600 resolution. Subtracted the mean and got the covriance of this matrix. Used the symmetric_eigen() function to generate the eigen values and multiplied this by the unrolled image matrix. We got the final reduced matrix of 1250 X 400 which we gave to SVM as the training model.

Passed each image in the test set to get the accuracy and got an accuracy of around 6%.

Section 2:

Function Used:

void train_test_haar(const Dataset &filenames, string value)
Following is the brief outline of the algorithm

Haar train:

- Step 0: Each Input image in the train folder is taken as an input image
- Step 1: Each input image is resized to dimensions 50x50
- Step 2: Random x(0-34), y(0-34) , height(1-8), width(1-6) values are generated
- Step 3: Respective rectangles formed and total sum of all pixel values inside the rectangle is calculated
- Step 4: For each (x, y) coordinate the pixel values below it is also calculated and sum of all pixels is taken
- Step 5: The absolute difference between them is calculated and values are stored as final features
- Step 6: 1000 values per image is given to the svm for training in the required format
  References: https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html.

  https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

- Format of the input train file used : target(1-25) feature : value (pairs) #info

Haar test:

- Step 0: Each Input image in the test folder is taken as an testing image
- Step 1: Each input image is resized (50 x 50)
- Step 2: Same steps as above
- Step 3: SVM multiclass classify package is used to classify using system()
  References: https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html.

- Format of the input train file used : target(1-25) feature : value (pairs) #info
- Accuracy achieved is 4.8%

Section 3:

Function Used:

1. void train(const Dataset & filenames)
2. void test_bow(const Dataset &filenames)
3. vector<float> kmean_process(vector<float> &mastermeans)
4. vector< vector<int> > assign_clusters(vector<float> means, const Dataset &filenames, int imagect)
5. vector<float> calc_newcentroids(vector<float> means, vector< vector<int> > assignedclusters,vector<float> mastermeans)

Following is the brief outline of the algorithm

Bag of words train:

- Step 0: Each Input image in the train folder is taken as an input image
- Step 1: The sift descriptors for each image is generated.
- Step 2: The mean for each of the descriptors is calculated and from the set of means a set of random k centroids are selected.
- Step 3: Each descriptor is than compared with k centroids and then they are assigned accordingly to each of the clusters wherever they have the closest related value with the k centroid.
- Step 4: Keep clustering till every cluster of descriptors containing the sift features are properly clustered with a mean value which represent the cluster most aptly.
- Step 5: For every image, get the representation for each cluster corresponding to the number of descriptors that have been assigned to it.
- Step 6: Create the k dimensional vector for the training file in which the count of the descriptors per image and cluster is written.
  References:
  https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html.
  http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html
- Format of the input train file used : target(1-25) feature : value (pairs) #info

Bag of words test:

a. Step 0: Each Input image in the train folder is taken as an input image
b. Step 1: The sift descriptors for each image is generated.
c. Step 2: The mean for each of the descriptors is calculated and from the set of means a set of random k centroids are selected.
d. Step 3: Each descriptor is than compared with k centroids and then they are assigned accordingly to each of the clusters wherever they have the closest related value with the k centroid.
e. Step 4: Keep clustering till every cluster of descriptors containing the sift features are properly clustered with a mean value which represent the cluster most aptly.

f. Step 5: For every image, get the representation for each cluster corresponding to the number of descriptors that have been assigned to it.
g. Step 6: Create the k dimensional vector for the test file in which the count of the descriptors per image and cluster is written.
References:
https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html.
http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html
h. Format of the input train file used : target(1-25) feature : value (pairs) #info

Analysis:

1. The accuracies of bow are below the baseline (ie. 20%).
2. The time taken to run the file takes a long time because the descriptors which are huge in number are pooled together and their cumulative mean is calculated.

2. DEEP FEATURES

Function Used:

void train_test_deep(const Dataset &filenames, string value)

Following is the brief outline of the algorithm

Deep train:

- Step 0: Each Input image in the train folder is taken as an input image
- Step 1: Each input image is resized to dimensions (231x231, 240x240, 250x250 etc)
- Step 2: The image is fed to overfeat application and the output is saved in deep.features file
  References: (http://cilvr.nyu.edu/doku.php?id=software:overfeat:start
- Step 3: The features are then passed to svm (total 4096 feature value pairs generated)
- Step 4: 4096 values per image is given to the svm for training in the required format
  References: https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html.
- Format of the input train file used : target(1-25) feature : value (pairs) #info

a. Deep test:

- Step 1: Each test image is resized to dimensions (231x231, 240x240, 250x250 etc)
- Using value 250x250 as it gives good accuracy
- Step 2: The image is fed to overfeat application and the output is saved in deep.features file
- References: (http://cilvr.nyu.edu/doku.php?id=software:overfeat:start
- Step 3: The features are then passed to svm classify(total 4096 feature value pairs generated)
- Step 4: 4096 values per image is given to the svm for testing in the required format
- References: https://www.cs.cornell.edu/people/tj/svm_light/svm_multiclass.html.

- Format of the input train file used : target(1-25) feature : value (pairs) #info

Analysis:

1. Time taken to train the file is very large (1hour) as compared to other approaches in part 1, 2
2. Accuracy 65.6% (larger than baseline 20%), 164 images were correctly detected.

```
[dipiband@tank ssurendr-cshelke-dipiband-a3]$ ./a3 test deep
Processing deep network model bagel : 1
Processing deep network model bread : 2
Processing deep network model brownie : 3
Processing deep network model chickennugget : 4
Processing deep network model churro : 5
Processing deep network model croissant : 6
Processing deep network model frenchfries : 7
Processing deep network model hamburger : 8
Processing deep network model hotdog : 9
Processing deep network model jambalaya : 10
Processing deep network model kungpaochicken : 11
Processing deep network model lasagna : 12
Processing deep network model muffin : 13
Processing deep network model paella : 14
Processing deep network model pizza : 15
Processing deep network model popcorn : 16
Processing deep network model pudding : 17
Processing deep network model salad : 18
Processing deep network model salmon : 19
Processing deep network model scone : 20
Processing deep network model spaghetti : 21
Processing deep network model sushi : 22
Processing deep network model taco : 23
Processing deep network model tiramisu : 24
Processing deep network model waffle : 25
Reading model...done.
Reading test examples... (250 examples) done.
Classifying test examples...done
Runtime (without IO) in cpu-seconds: 0.15
Average loss on test set: 34.4000
Zero/one-error on test set: 34.40% (164 correct, 86 incorrect, 250 total)
prediction file generated...
Accuracy of Deep Classifier:: 65.6%
```

TIME Analysis

Sr. No.	Model file Name	Model file generation After reading training samples (Time in sec)	Total time for training
1	model_file_svm	486s	9 mins
2	model_file_haar	160s	4 mins
3	model_file_deep	72.1	1 hour
4	model_file_pca	720s	12 mins
5	model_file_bow	59	1 hour