ECS797-Machine Learning for Visual Data Analysis

LAB1: Image Classification using a Bag of Words Model

Bhanu Pratap Singh (210760006)

Id- ec21279@qmul.ac.uk

Image classification is the process of categorizing and labelling groups of pixels or vectors within an image based on specific rules. The categorization law can be devised using one or more spectral or textural characteristics. Two general methods of classification are 'supervised' and 'unsupervised'. Here we are using Bag of words Model for image classification, which is a vector of occurrence counts of a vocabulary of local image features.

1.Getting Started

Downloading the file and extracting the content in it. Checking the presence of all three sub folders in the main file. Here our software is given on Matlab and is dependent on concerning dependencies (libvsm-3.14, vlfeat-0.9.16). In this part we have added the software path to our workspace.

Code used:

```
addpath(genpath('software'));
```

2.Dictionary Creation- Feature Quantization

1. Executing the command in file Lab1.m to load the pre-computed features for the training and testing images. After execution two variable TrainMat of dimensions 270000*128 and TestMat of dimensions 90000*128 are loaded. Also the sift feature has the dimensions of 128 .
   Code used:
   ```
   load('data/global/all_features');
   ```

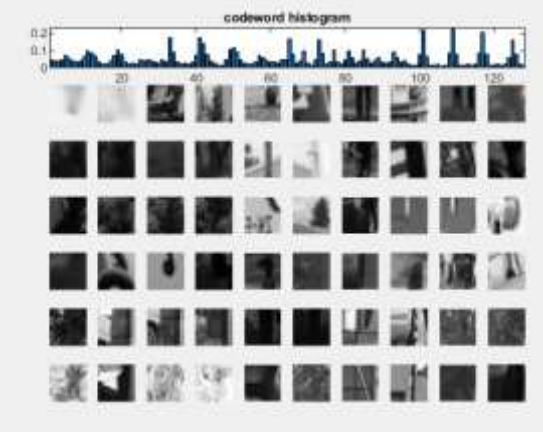
2. Creating the dictionary by clustering a subset of extracted descriptors and using the dictionary of 500 words is created using the k-means clustering. The size of the dictionary and size of the features is controlled by K-means clusters. After running the code variable C of size 500*128 will be saved in the dictionary .mat
   Code used:
   ```
   save('data/global/dictionary','C');
   ```

3. Calculating the Euclidean distance between image descriptors and codewords or clusters centres. There is function named EuclideanDistance.m defined in the file which calculated the Euclidean distance.
   Code Used:
   ```
   %% 2.3 Euclidean distance
   % Assume a and b are two vectors, the  Euclidean distance function is EuclideanDistance.m
   %-----------------------------------code-------------------------------
   %{%
   clear;
   load('data/global/all_features');
   load('data/global/dictionary');
   a = TestMat(1,:);
   b = C(1,:);
   d = EuclideanDistance(a,b);
   %}
   %---------------------------------end of code ---------------------------
   ```
   Output:
   Euclidean Distance d = 1.0036

4. Assigning each descriptor in the testing and training images to the nearest codeword clusters by using the min() function. Here we are assigning index_train as assignment vector of training images of size 1*270000 and index_test as assignment vector of testing image of size 1*90000 and saving them.
   Code used:

```
discrptor_test = TestMat(:,:);
d = EuclideanDistance(TestMat,C);
[minv,index_test] = min(d ,[], 2);% index will be the nearest codeword cluster
index_test = index_test';

discrptor_train = TrainMat(:,:);
e = EuclideanDistance(TrainMat,C);
[minv1,index_train] = min(e , [], 2);
index_train = index_train';


save('data/global/assignd_discriptor','index_train','index_test');
%)
```

5. Visualisation of image patches which are assigned to same codeword using the function visualize_patches.m
   Output:

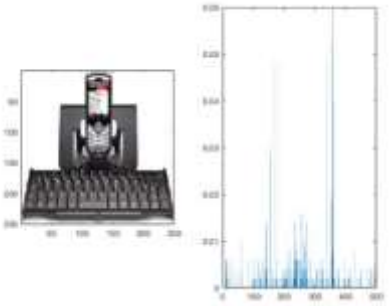


3. Image representation using Bag of words representation

1. Representing each image in the training and the test dataset as a histogram of visual words which represent each image using the Bag of Words representation and normalising the histogram by their L1 norm. we are normalising by calling the function do_normalise() which is in the file do_normalise.m in the software directory. The feature data for each image is read which is stored in data/local/*ID*. Both testing and the training dataset images are stored in the same matrix called BoW in which training images are the entries from 1 to 300 and the testing images are the entries from the 301 to 400. Therefore , the matrix BoW is 400*500 matrix which have been saved.
   Code used:

```
BoW =[]; %initialization
isshow = 0; % show image and histogram or not
load('data/global/image_names');
load('data/global/dictionary','C');
load('data/global/all_features');
load('data/global/assignd_discriptor');
nimages = 400;
vocbsize = 500;
DictionarySize=500;
for ii = 1:nimages
    image_dir=sprintf('%s/%s/','data/local',numstring(ii,3));          % location where detector is sa
    infName = fullfile(image_dir, sprintf('%s', 'sift_features'));
    load(infName, 'features');

    %---------------------- write your own code here-----------------------------------
    %---------------------- write your own code here-----------------------------------
    d=EuclideanDistance(features.data, C);
    [mind1,index] = min(d');
    histo = histcounts(index, DictionarySize);
    BoW(ii,:) = do_normalize (histo);

    if isshow == 1
      close all; figure;
      subplot(1,2,1),subimage(imread(strcat('image/',image_names{ii})));
      subplot(1,2,2),bar(BoW(:,ii)),xlim([0 500]);
    end
end

%
save('data/global/bow','BoW')
```

Output:



4. Image classification using Nearest Neighbour (NN) classifier

1. Applying the steps of NN classifier as described in lab1.m and calculating the L2 euclidean distance between the test and the training image and assigning to each test the label of its nearest neighbour in the training test in which we are assigning each test image to the class of its nearest neighbour in the training set.

   The arguments of this are:

   test_data: TxD matrix. T is test samples with D dimensional features

   train_data: NxD, N is training samples with D dimensional features

   k: 'k' of KNN algorithm

   method: distance method option, 1 for L2 distance, 2-historgram inspection.

   Output:

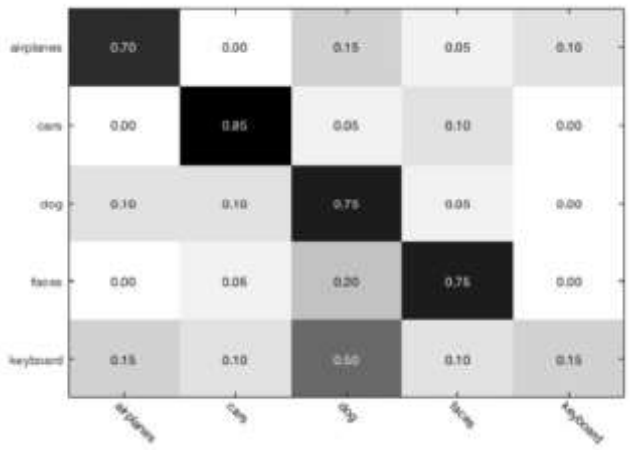
Name	Value
k	1
method	1
NNresult	*100x1 double*
predict_label	*100x1 double*
tem	*20x1 double*
test_data	*100x500 double*
test_labels	*100x1 double*
train_data	*300x500 double*
train_labels	*300x1 double*

2. Computing and reporting the overall classification error per class.
   Output:

Class	error
1	0.1500
2	0.1500
3	0.5500
4	0.0500
5	0.7500

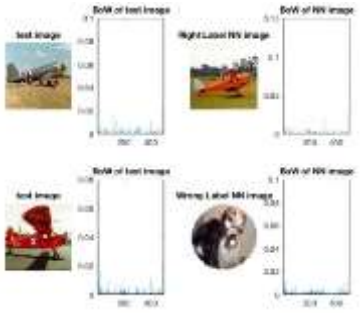
   Overall mean error =0.3300

3. Computing the confusion matrix. Confusion matrix is matrix that compares the actual target values with those predicted by the machine learning model.
   Output:

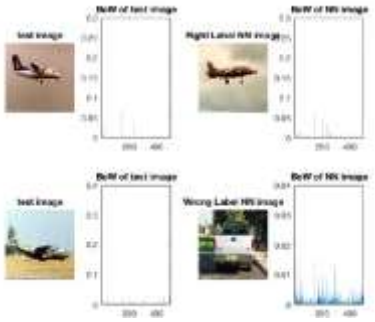


4. For each class showing some images that are correctly classified and some images that are incorrectly classified

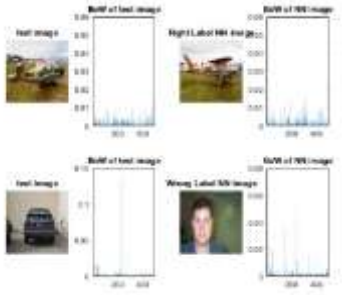
Output:



Number of correct = 2 Number of wrong = 1



Number of correct = 1 Number of wrong =2



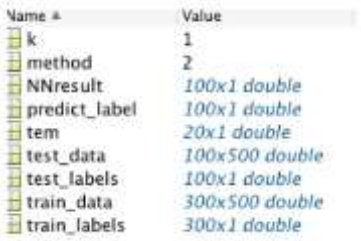
Number of correct = 5 Number of wrong = 5

5. Computing the histogram intersection between two histograms by using the method =2 in section 4.1 and implementing histogram intersection code in the knnsearch.m

Code:

```
function d=histogram_intersection(a,b)
  p= size(a,1);
  p=size(a,2); % dimension of samples

  assert(p == size(b,2)); % equal dimensions
  assert(size(a,1) == 1); % a needs to be a single sample
  assert(size(b,1) == 1); % b needs to be a single sample

  %d=zeros(m,1); % initialize output array


  d = 0;
  % -------------- write your own code here ---------------
  d = 1 - sum(min(a,b));
  % -------------- write your own code here ---------------

end
```

Output:



Errors:

Class	Error
1	0.1000
2	0.2000
3	0.3500
4	0.2500
5	0.5000

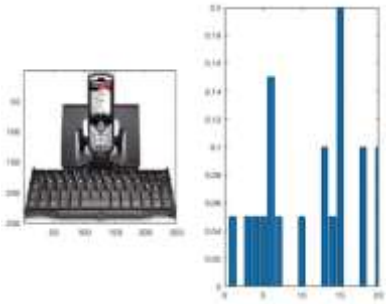
Overall mean error = 0.2800

Analysis:
If we compare our L1 method in which we are getting overall error =0.3300 with the method 2 which is histogram intersection in which we are getting error = 0.2800 means we are when we are doing the histogram intersection, we are getting the better visualisation and better accuracy.

5.Dictionary size

1. Performing the classification experiment using a very small dictionary and report the classification error and confusion matrices. Using the dictionary size =20 and here the size of our BoW is 400*20.

Output:



Errors:

When using method 1 we get the following error and dictionary size = 20

Class	Errors
1	0.0500
2	0.1000
3	0.6000
4	0.3500
5	0.6500

Mean overall error = 0.3500

When using the histogram intersection method by using method = 2

Class	Error
1	0.0500
2	0.1000
3	0.5500
4	0.2000
5	0.5500

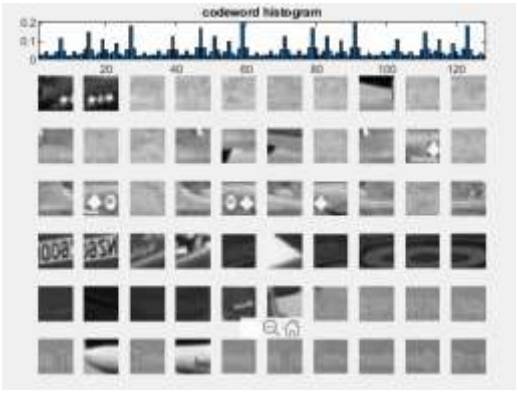
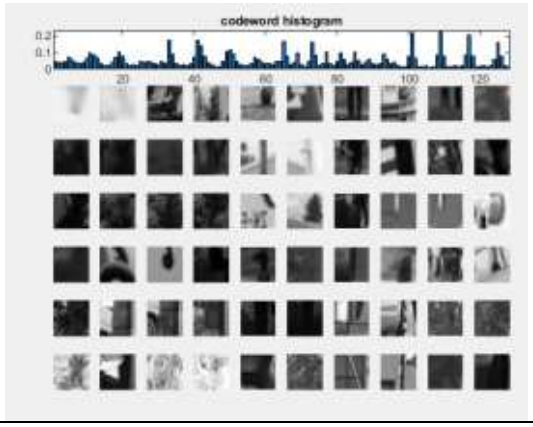
Mean overall error = 0.2900

Analysis:

If we compare with the small dictionary size our error got increased by 2 %. Also, if we compare between method 1 and method 2 we analysed that error reduced by 6% when we opt for the histogram intersection using method 2.

2. We can see that there is drop in performance by when we change the dictionary size from 500 to 20. We notice by using the method 1the error has increased by 2% and also when we opt for the histogram intersection method using method 2 there is 2% increase in the error which implies our performance gets reduced when we minimise the dictionary size. It is hard to give a high accuracy with such less code vector for such a large dataset. Therefore large dictionary size must be used for better visualisation. Repeating the step 2.5 for better visualisation.

Output :

Dictionary size=20, wordID=12 , n=30	Dictionary size=500, wordID=12 , n=30
	

6. Image classification using Support Vector Machine(SVM) classifier

1. Training the linear multiclass SVM classifier for each of the image classes by using the libsvm library. By uncommenting the code in the 6.1 and using the dictionary size of 500 for getting best c , g value and the cross validation accuracy.

Code Used:

```
clear;
load('data/global/bow','Bow');
fprintf('\nClassification using SVM ref_svm\n');
tem = ones(60,1);
train_labels  = [tem;2*tem;3*tem;4*tem;5*tem];
train_data    = Bow(1:300,:);
tem = ones(20,1);
test_labels   = [tem;2*tem;3*tem;4*tem;5*tem];
test_data     = Bow(301:400,:);
clear Bow;
% set the parameters via cross-validation! Elapsed time is 166.922776 seconds.
bestc=64;bestg=2.2974;
bestcv=0;
% t1;
% for log2c = -1:10,
%   for log2g = -1:0.1:2.5,
%     cmd = ['-v 5 -t 2 -c ', num2str(2^log2c), ' -g ', num2str(2^log2g)];
%     cv = svmtrain(train_labels, train_data, cmd);
%     if (cv >= bestcv),
%       bestcv = cv; bestc = 2^log2c; bestg = 2^log2g;
%     end
%     fprintf('%g %g %g (best c=%g, g=%g, rate=%g)\n', log2c, log2g, cv, bestc, bestg, bestcv);
%   end
% end
% toc
options=sprintf('-s 0 -t 2 -c %f -b 1 -g %f -q',bestc,bestg);
model=svmtrain(train_labels, train_data,options);
```

Output:

- Cross validation accuracy = 73.86%
- best c = 127 best g =1.2946 rate = 82.23

2. Applying the linear SVM classifier to the testing images and classifying them and the accuracy is calculated by running the section 6.2

   Code:

```
%{%
[predict_label, accuracy , dec_values] = svmpredict(test_labels,test_data, model,'-b 1');
%}
```

   Output:

   Accuracy = 77.83%

3. Calculating the overall and classification error per class.

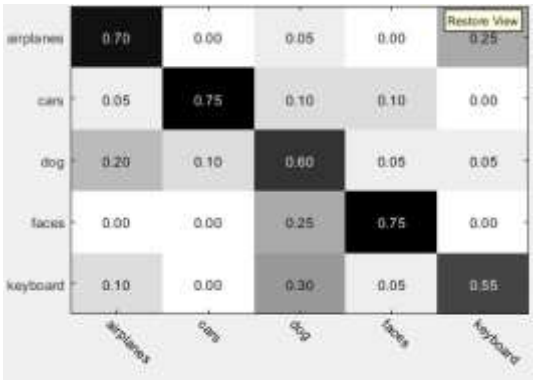
   Output:

Class	Error
1	0.07
2	0.07
3	0.32
4	0.32
5	0.48

Mean overall error = 0.252

4. Computing and showing the confusion matrix

   Output:



5. Showing the images that are correctly classified and that are incorrectly classified.
Output: