# ECS797 Machine Learning for Visual Analysis

## Lab 1: Image Classification using a Bag-of-Words model

**Name : Tanmaiyii Rao**
**ID : 140361229**

## 1. Getting started

This coursework is using Bag of Words Model for Image Classification. As given in the code template, the software path has been loaded and the concerning dependencies (vlfeat - -9.16 and libsvm-3.14 have been loaded).

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

## 2. Dictionary creation – feature quantization

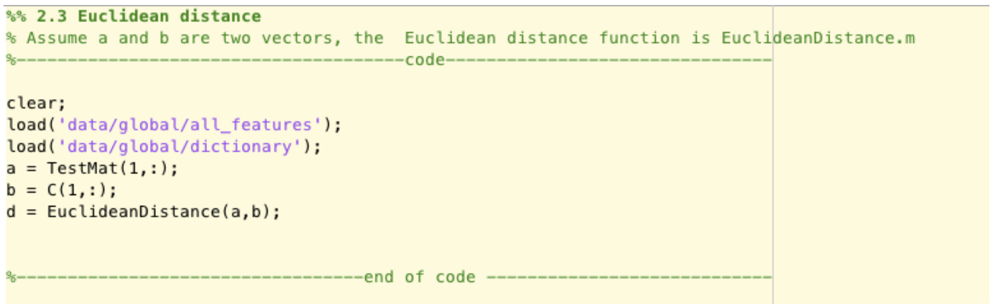
1. The commands in lab1.m have been executed to load the pre-computed features for the training and test images. The variables, namely, TrainMat (90000x128) and TestMat (2700x128), are loaded. The sift feature has 128 dimensions. The following code has been executed.

   ```
   load('data/global/all_features')
   ```

2. The code in 2.2 has been run, and a dictionary of 500 words has been created by k-means clustering a subset of the extracted descriptors. The number of kmeans clusters controls the size of the dictionary and the size of the features. After running the code the dictionary has been saved as C. The variable C , a 500x128 matrix will be saved in *dictionary.mat* .

   ```
   save('data/global/dictionary','C');
   ```

3. The Euclidean distance between image descriptors and codewords (i.e cluster centres) has been calculated in this section. The function file that does this is EuclideanDistance.m. The following code has been run in lab1.m

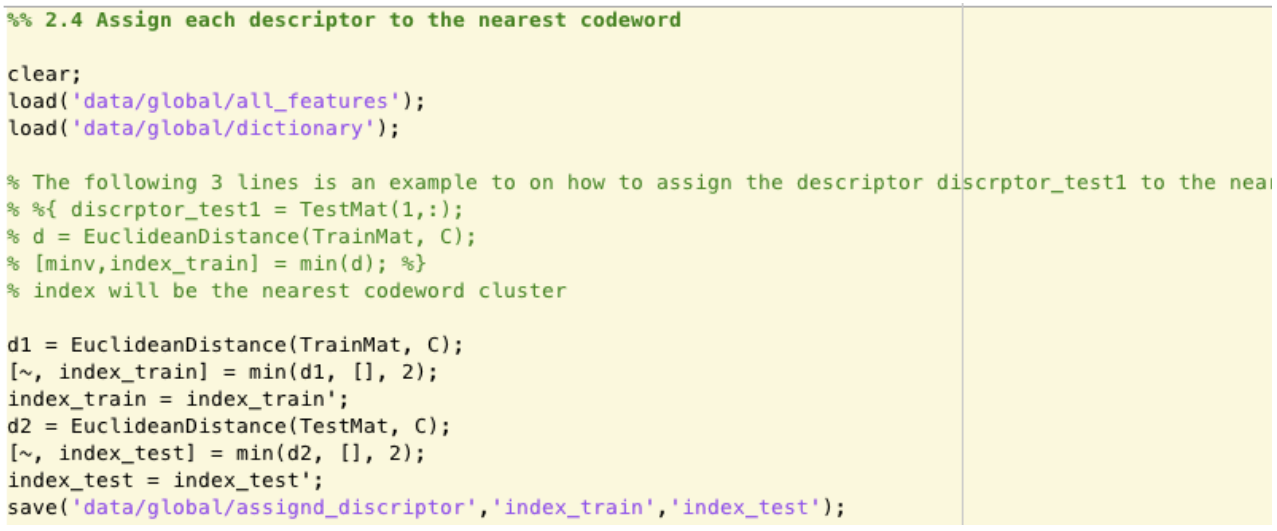


```
%% 2.3 Euclidean distance
% Assume a and b are two vectors, the  Euclidean distance function is EuclideanDistance.m
%--------------------------------------code--------------------------------

clear;
load('data/global/all_features');
load('data/global/dictionary');
a = TestMat(1,:);
b = C(1,:);
d = EuclideanDistance(a,b);


%----------------------------------end of code ---------------------------
```

The Euclidean Distance d = 1.0036

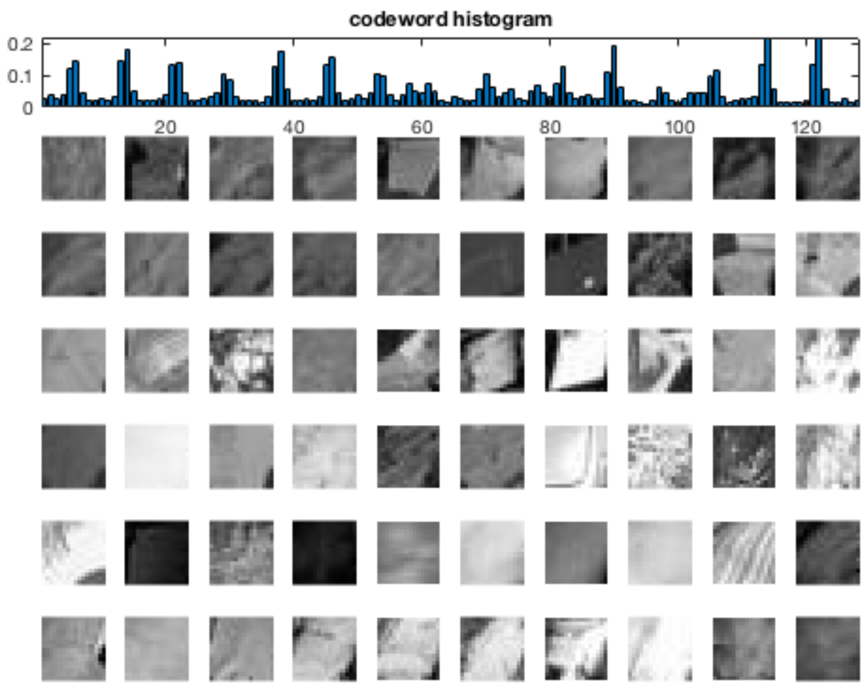
.

4. Here, I've written a code that assigns each descriptor in the training and test images to the nearest codeword cluster. To compute this, the min() function has been used. The assignment vector of training images is index_train and that of test images is index_test. The dimensions are set to be 270000x1 and 90000x1 respectively. The following code has been implemented. As can be seen in the image, the transpose has been taken to make the dimensions to be set the way described in the question i.e 1x270000 and 1x90000.



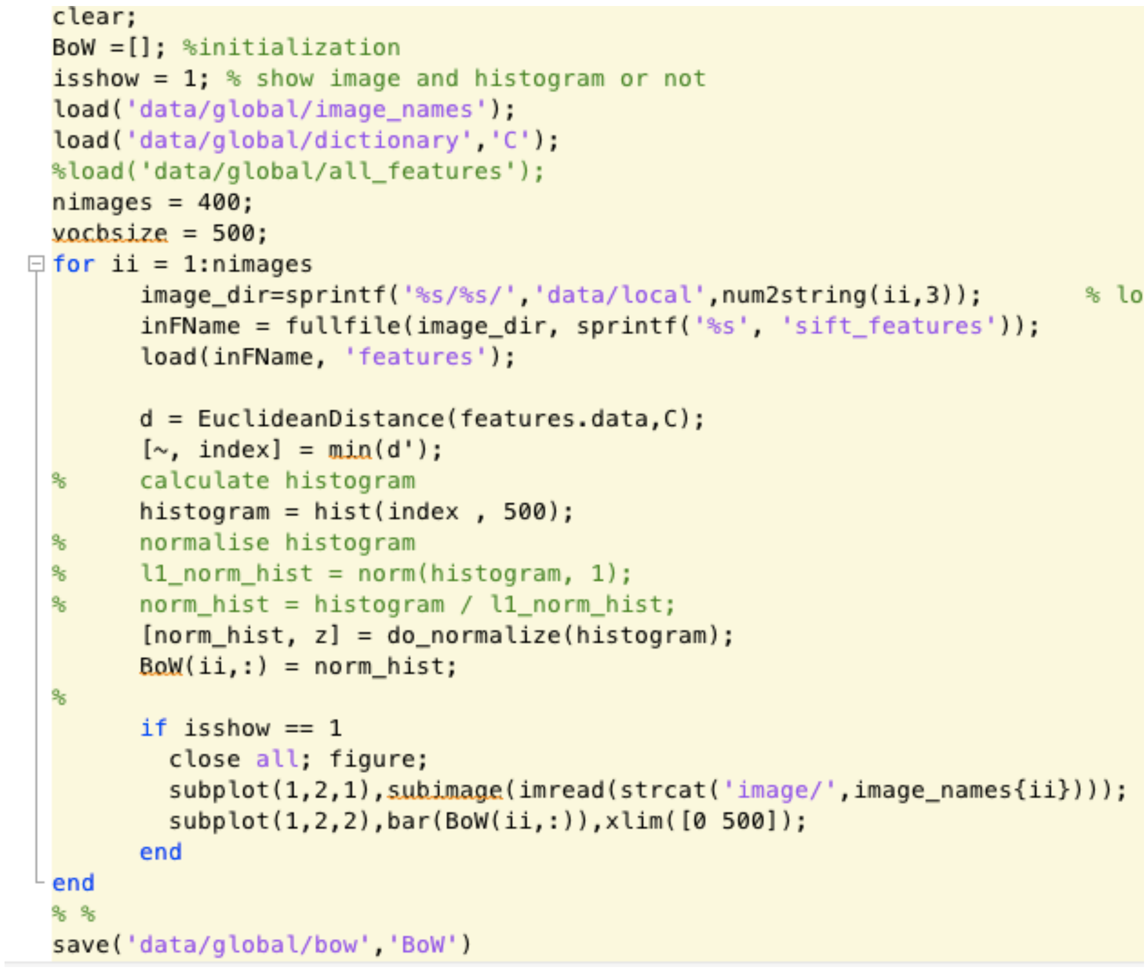
```
%% 2.4 Assign each descriptor to the nearest codeword

clear;
load('data/global/all_features');
load('data/global/dictionary');

% The following 3 lines is an example to on how to assign the descriptor discrptor_test1 to the nea
% %{ discrptor_test1 = TestMat(1,:);
% d = EuclideanDistance(TrainMat, C);
% [minv,index_train] = min(d); %}
% index will be the nearest codeword cluster

d1 = EuclideanDistance(TrainMat, C);
[~, index_train] = min(d1, [], 2);
index_train = index_train';
d2 = EuclideanDistance(TestMat, C);
[~, index_test] = min(d2, [], 2);
index_test = index_test';
save('data/global/assignd_discriptor','index_train','index_test');
```

5. The image patches that are assigned to the same codeword (the code computed above) have been visualised. The function file to visualise the patches is visualize_patches.m

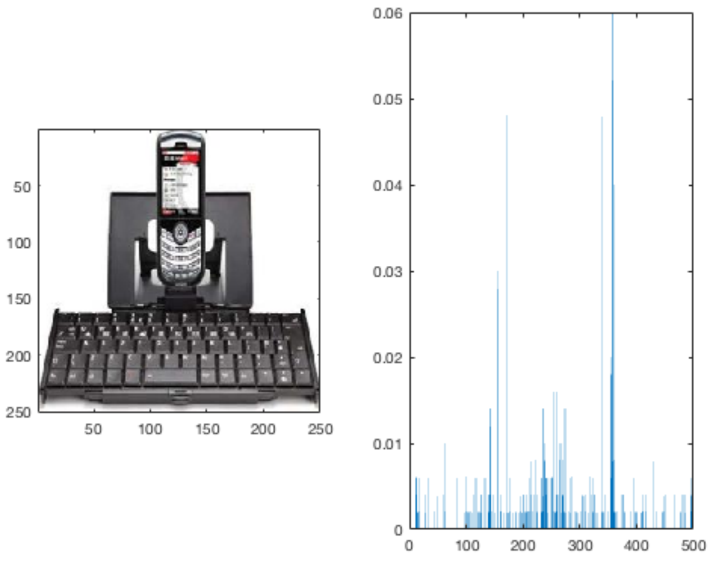


## 3. Image Representation using Bag of Words Representation

1. Each image in the training and test dataset have been represented as a histogram of visual words. The histograms have been normalised using the L1 norm. The do_normalise.m function file has been used to compute this. The standard way where you calculate L1 norm and normalisation after, has also been used which has been commented out. Both the methods yield the same results. The feature data for each image is read  which is stored in data/local/*ID*. The images of both the training and testing dataset have been stored in the same matrix called BoW. The entries from 1 to 300 are representations of the training images and the entries from 301 to 400 are representations of the test images. Therefore, the matrix BoW is 400x500 matrix which has been saved. The following code has been implemented to calculate the BoW and the visualisation of BoW.



```
clear;
BoW =[]; %initialization
isshow = 1; % show image and histogram or not
load('data/global/image_names');
load('data/global/dictionary','C');
%load('data/global/all_features');
nimages = 400;
vocbsize = 500;
for ii = 1:nimages
      image_dir=sprintf('%s/%s/','data/local',num2string(ii,3));        % lo
      inFName = fullfile(image_dir, sprintf('%s', 'sift_features'));
      load(inFName, 'features');

      d = EuclideanDistance(features.data,C);
      [~, index] = min(d');
%     calculate histogram
      histogram = hist(index , 500);
%     normalise histogram
%     l1_norm_hist = norm(histogram, 1);
%     norm_hist = histogram / l1_norm_hist;
      [norm_hist, z] = do_normalize(histogram);
      BoW(ii,:) = norm_hist;
%
      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')
```

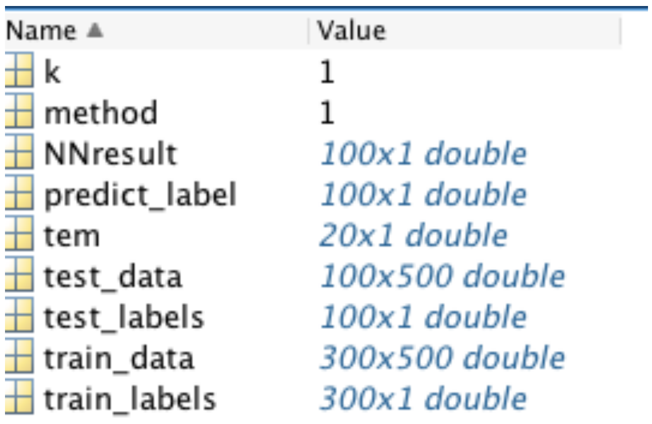


## 4. Image classification using a Nearest Neighbour (NN) classifier

1. The steps of the NN classifier as described in the lab1.m scripts have been run accordingly in section 4.1. The L2 distances between the test and the training images have been calculated and each test image has been given the label of its nearest neighbour in the training set (i.e each test image has been assigned to the class of its nearest neighbour in the training set). The input arguments of this are:

- test_data -> TxD matrix, T test samples with D dimensional features.
- train_data -> NxD matrix, N training samples with D dimensional features
- k -> ‘k’ of KNN algorithm
- method -> distance method option, 1-L2 distance, 2-histogram intersection. The method used in this section is method 1.

  The output is:
- NNresult -> Txk matrix, the k-NNs for each of the T samples.



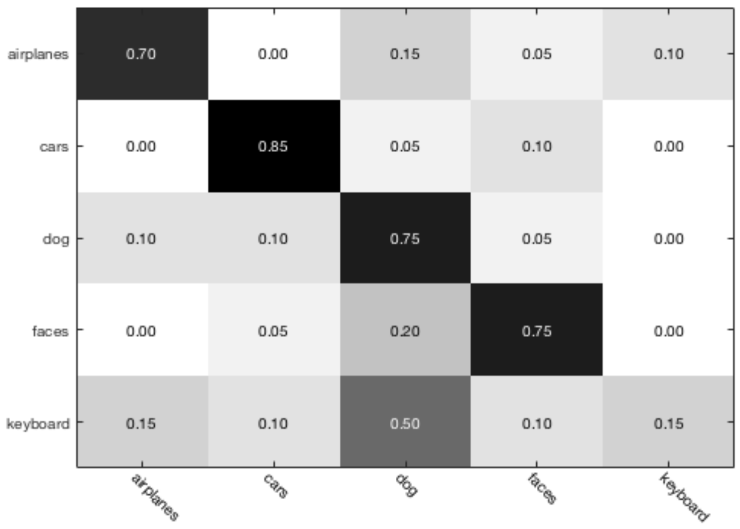
| Name ▲ | Value |
|---|---|
| k | 1 |
| method | 1 |
| NNresult | *100x1 double* |
| predict_label | *100x1 double* |
| tem | *20x1 double* |
| test_data | *100x500 double* |
| test_labels | *100x1 double* |
| train_data | *300x500 double* |
| train_labels | *300x1 double* |

2. The code in section 4.2 in lab1.m has been run. The overall and the classification errors per class have been computed and reported. These errors are for method 1.

| Class | Error/class |
|---|---|
| 1 | 0.30 |
| 2 | 0.15 |
| 3 | 0.25 |
| 4 | 0.25 |
| 5 | 0.85 |

| | |
|---|---|
| Mean overall error | 0.36 |

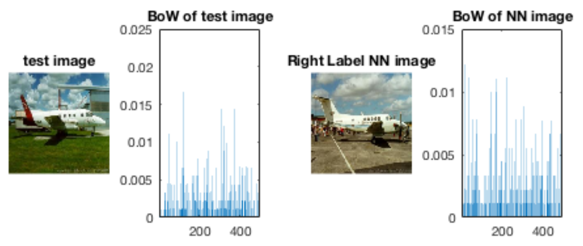
3. The confusion matrix is computed by running the code in 4.3 of lab1.m.

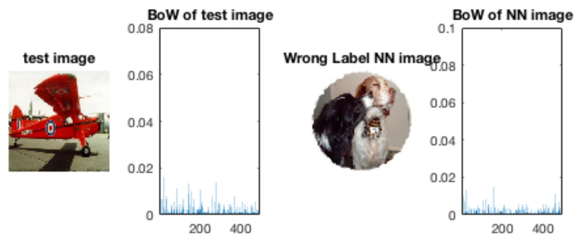


| | airplanes | cars | dog | faces | keyboard |
|---|---|---|---|---|---|
| airplanes | 0.70 | 0.00 | 0.15 | 0.05 | 0.10 |
| cars | 0.00 | 0.85 | 0.05 | 0.10 | 0.00 |
| dog | 0.10 | 0.10 | 0.75 | 0.05 | 0.00 |
| faces | 0.00 | 0.05 | 0.20 | 0.75 | 0.00 |
| keyboard | 0.15 | 0.10 | 0.50 | 0.10 | 0.15 |

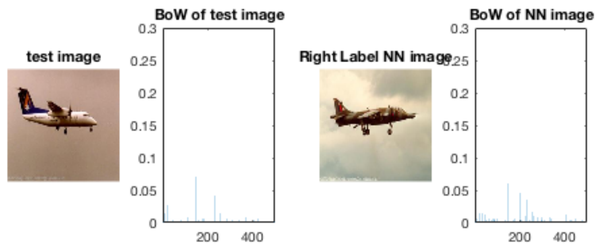
4. For each class, some images that are correctly classified and some images that are incorrectly classified are visualised. The number of correctly and incorrectly classified images is set.

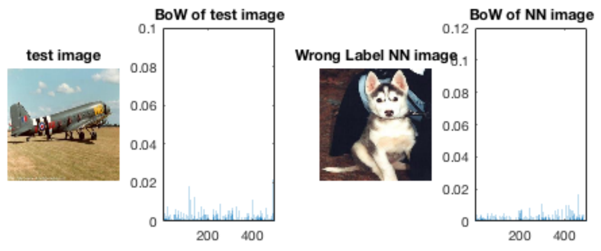
   No. of Correct = 2 , No. of Wrong = 1



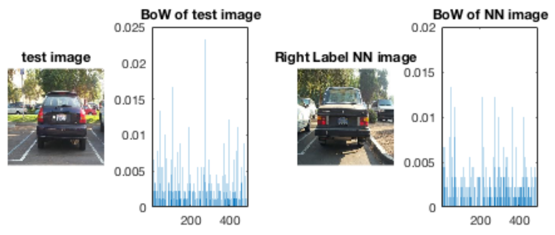


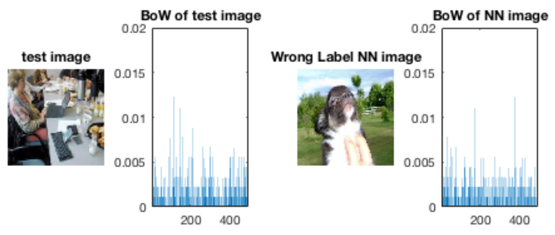
   No. of Correct = 1 , No. of Wrong = 2



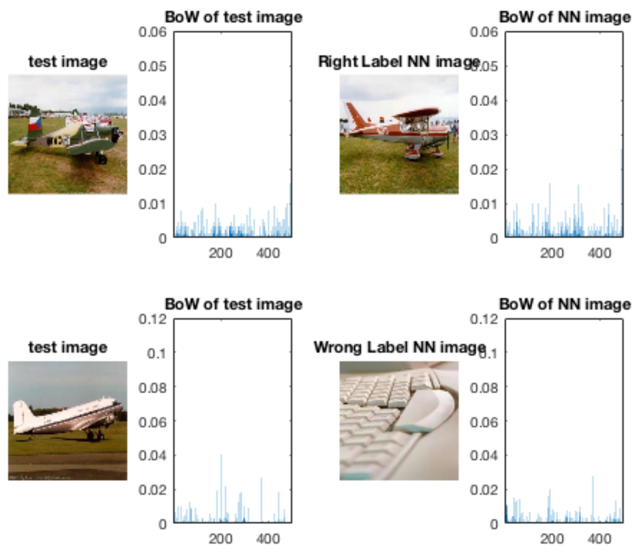


   No. of Correct = 20 , No. of Wrong = 25

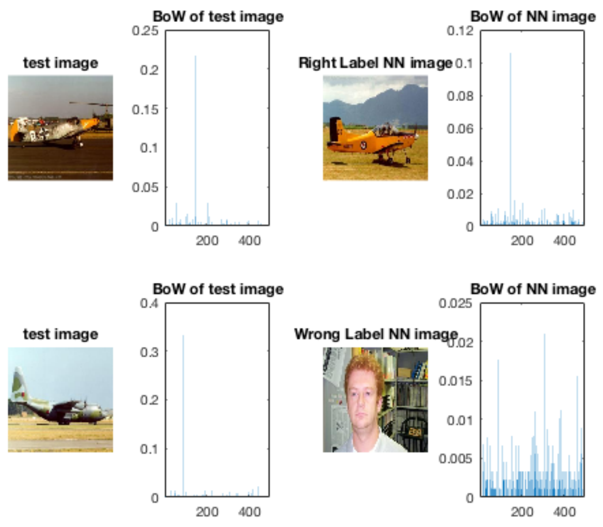




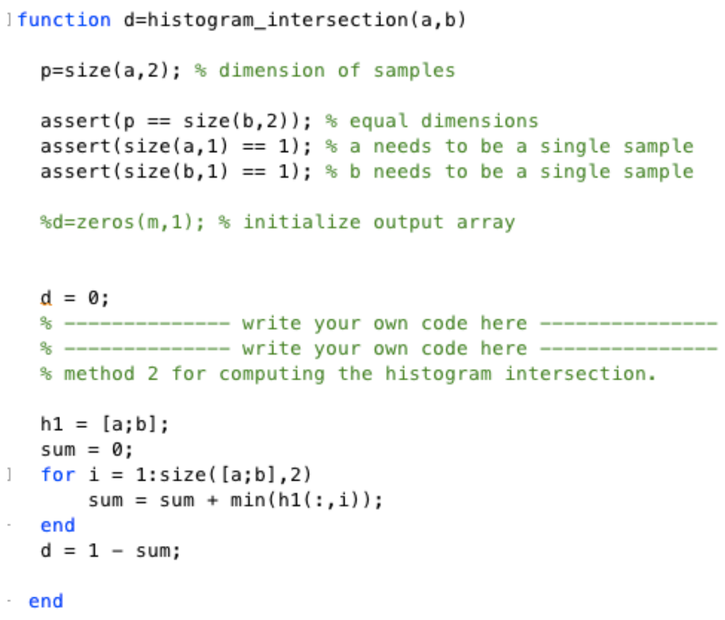
No. of Correct = 3 , No. of Wrong = 3



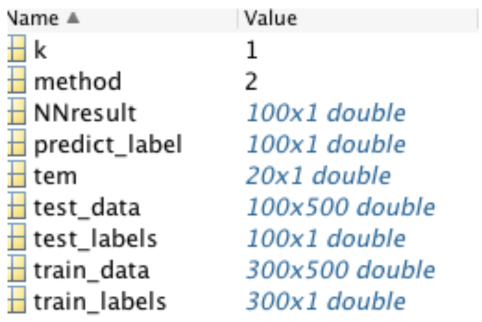
No. of Correct = 10 , No. of Wrong = 5



5. The code for computing the histogram intersection between two histograms has been computed by using method 2. The code for computing the histogram intersection method 2 has been written in the file knnsearch.m . All the above steps have been implemented again with method 2.



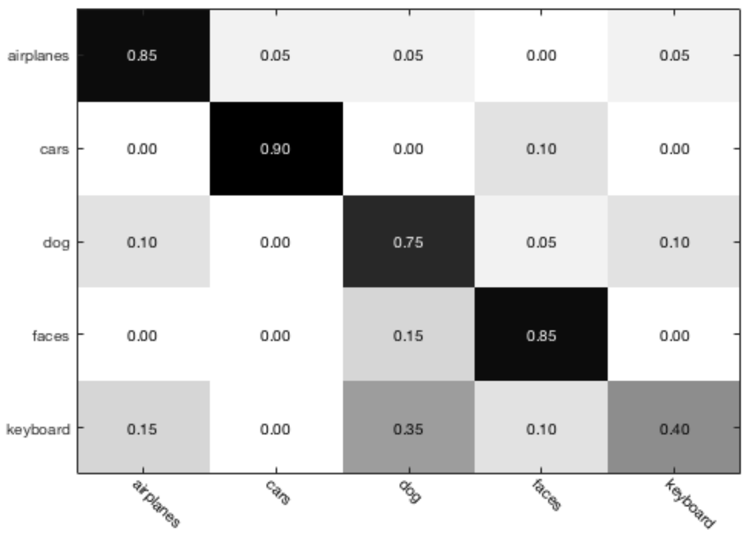
```
function d=histogram_intersection(a,b)

  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 ---------------
  % -------------- write your own code here ---------------
  % method 2 for computing the histogram intersection.

  h1 = [a;b];
  sum = 0;
  for i = 1:size([a;b],2)
      sum = sum + min(h1(:,i));
  end
  d = 1 - sum;

 end
```



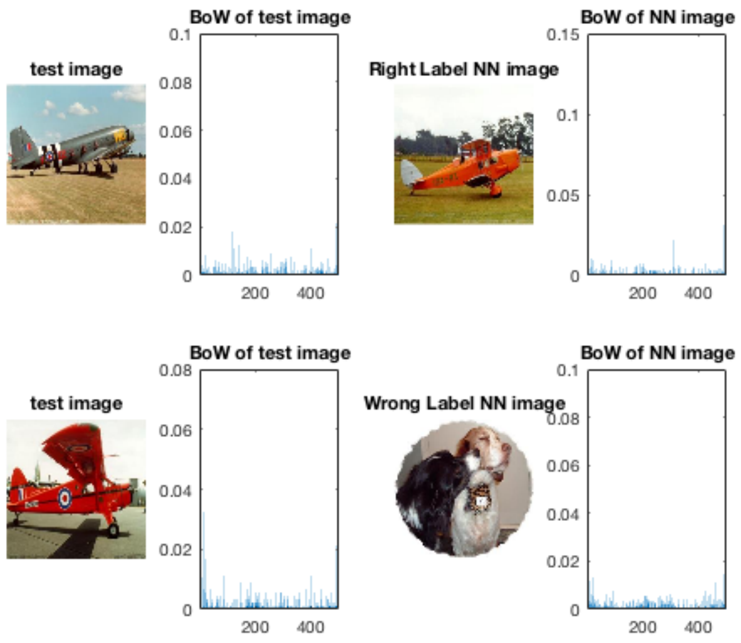
| Name ▲ | Value |
|---|---|
| k | 1 |
| method | 2 |
| NNresult | 100x1 double |
| predict_label | 100x1 double |
| tem | 20x1 double |
| test_data | 100x500 double |
| test_labels | 100x1 double |
| train_data | 300x500 double |
| train_labels | 300x1 double |

| Class | Error/Class |
|---|---|
| 1 | 0.15 |
| 2 | 0.10 |
| 3 | 0.25 |
| 4 | 0.15 |
| 5 | 0.60 |

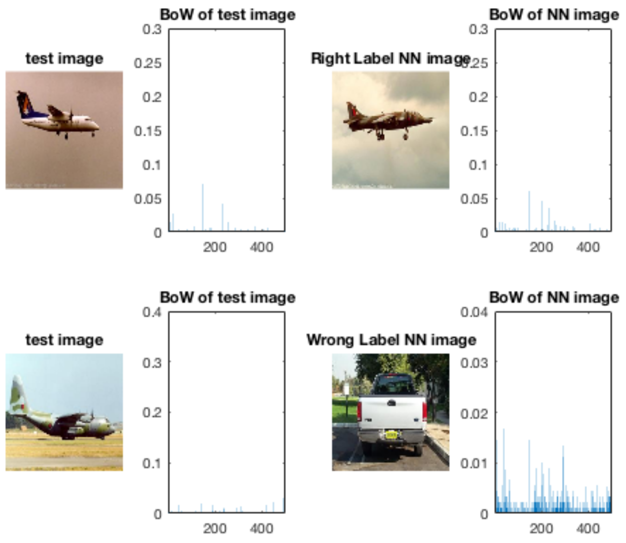
| Mean overall error | 0.25 |
|---|---|



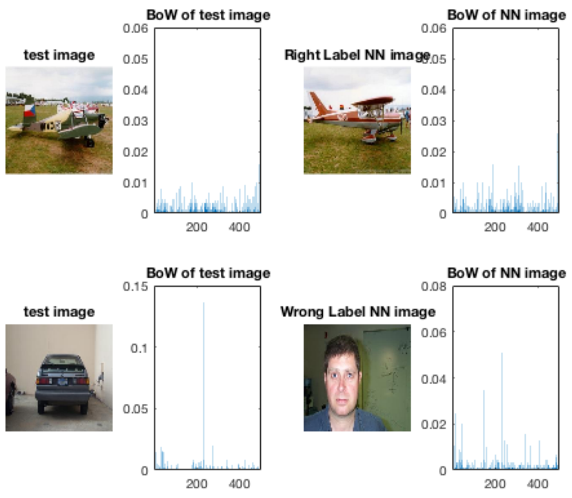
No. of Correct = 2 , No. of Wrong = 1



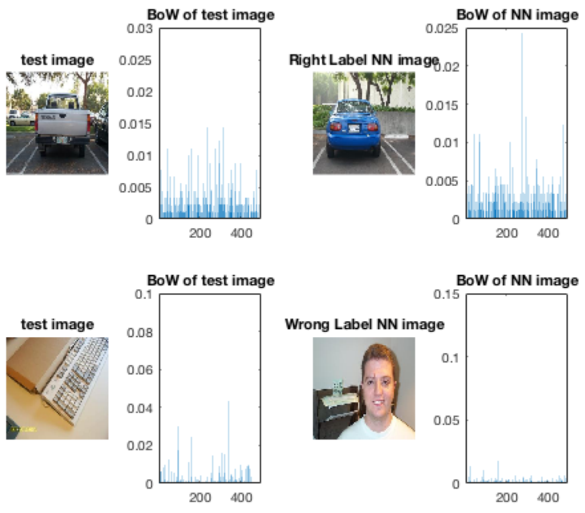
No. of Correct = 1 , No. of Wrong = 2



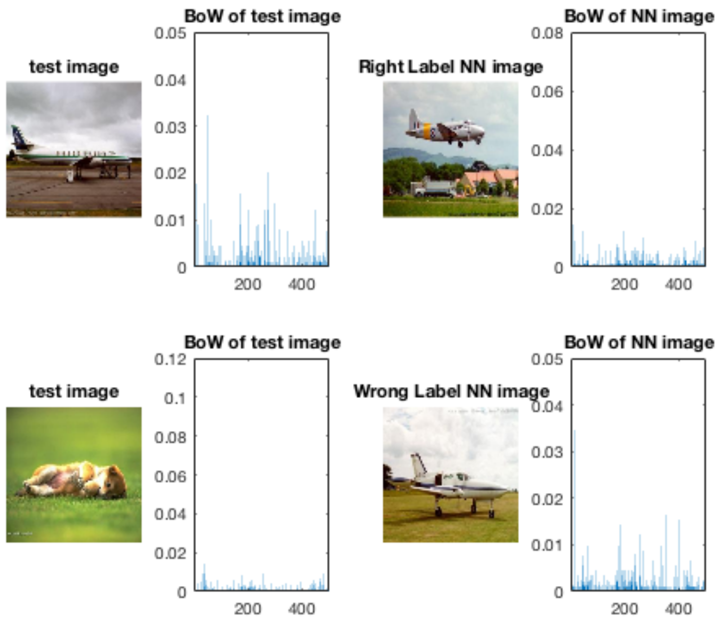
No. of Correct = 5 , No. of Wrong = 5



No. of Correct = 20 , No. of Wrong = 15

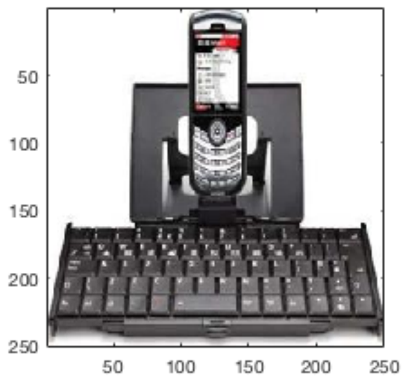


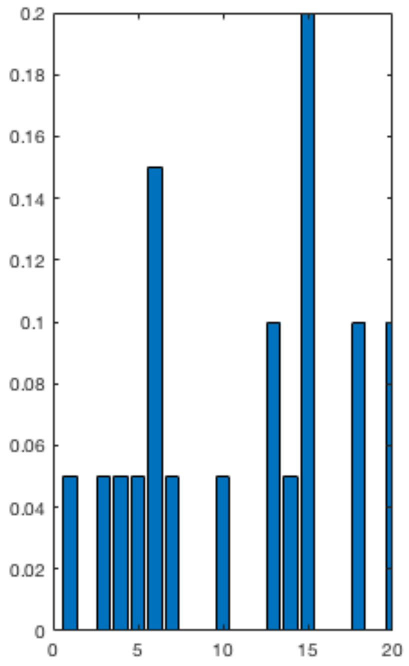
No. of Correct = 6 , No. of Wrong = 10



## 5. Dictionary Size

1. The classification experiment has been performed using a very small dictionary ( size = 20) . All the steps above have been repeated the classification error and the confusion matrices have been reported. The BoW will now have a size 400 x 20. Below the graphs and errors for method 1 and 2 have been reported.





Method 1 errors

| Class | Error/Class |
|---|---|
| 1 | 0.70 |
| 2 | 0.80 |
| 3 | 0.75 |
| 4 | 0.75 |
| 5 | 0.80 |

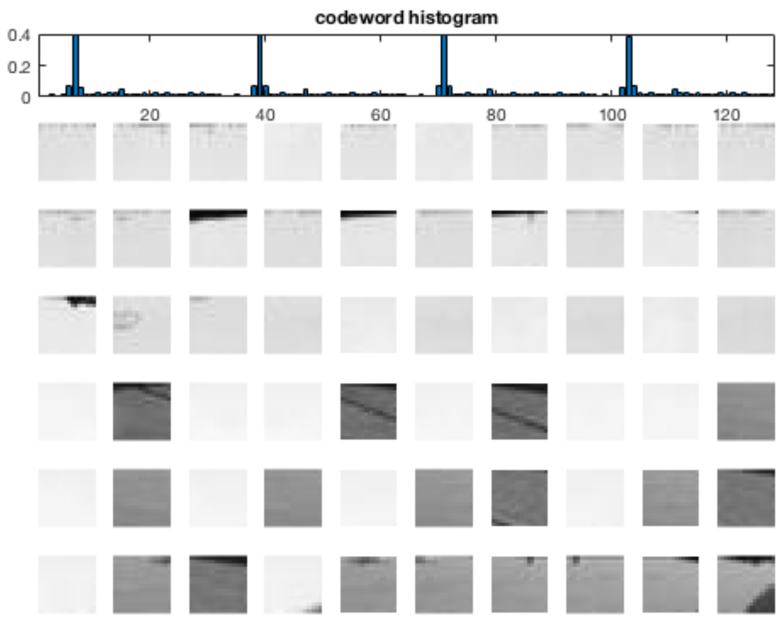
| Mean overall error | 0.76 |
|---|---|

Method 2 errors

| Class | Error/Class |
|---|---|
| 1 | 0.60 |
| 2 | 0.80 |
| 3 | 0.80 |
| 4 | 0.65 |
| 5 | 0.80 |

| Mean overall error | 0.73 |
|---|---|

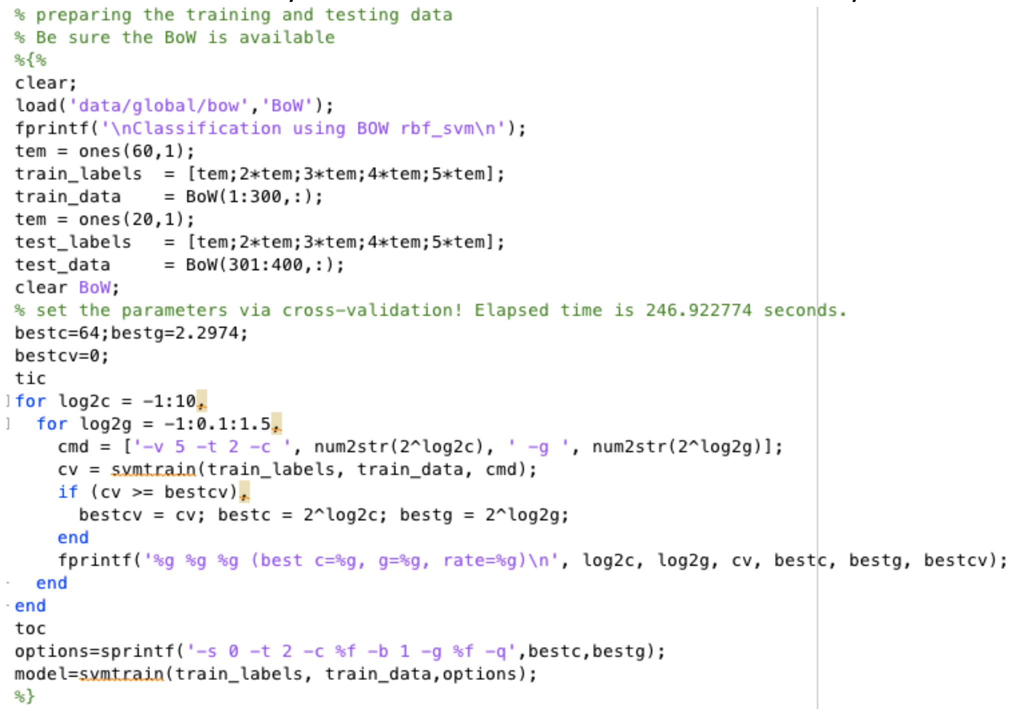
2. It can be seen that there is a large drop in performance, with the a significant increase in classification error. This is because the size of the dictionary is too small compared to the size of the test and train data. It is hard to give a high accuracy with such less codevectors for such a large dataset. Step 2.5 has been repeated to prove this and the codevectors histogram is visualised.



As can be seen in the histogram, there are more repetitive patches of the same kind in the whole codebook.

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

1. The code in 6.1 has been run to train a linear multiclass SVM for each of the image classes. The following code was uncommented and the procedure that calculates the optimal values for the parameter C using cross validation was observed. You could tell that the accuracy for C.V increases for every iteration. The BoW was used for dictionary size = 500.

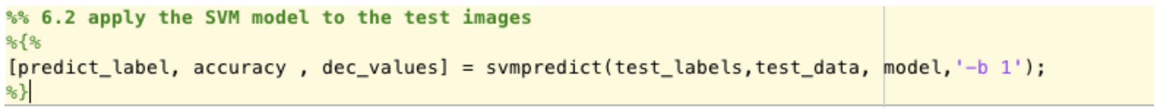


```
% preparing the training and testing data
% Be sure the BoW is available
%{%
clear;
load('data/global/bow','BoW');
fprintf('\nClassification using BOW rbf_svm\n');
tem = ones(60,1);
train_labels  = [tem;2*tem;3*tem;4*tem;5*tem];
train_data    = BoW(1:300,:);
tem = ones(20,1);
test_labels   = [tem;2*tem;3*tem;4*tem;5*tem];
test_data     = BoW(301:400,:);
clear BoW;
% set the parameters via cross-validation! Elapsed time is 246.922774 seconds.
bestc=64;bestg=2.2974;
bestcv=0;
tic
for log2c = -1:10,
  for log2g = -1:0.1:1.5,
    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);
%}
```

The following results were obtained after running this code:

- Cross Validation Accuracy = 81.3333%
- 10 1.5 81.3333 (best c=128, g=1.31951, rate=81.6667)

2. The linear SVM classifier has been applied to the test images and the test images have been classified and the accuracy has been reported.



```
%% 6.2 apply the SVM model to the test images
%{%
[predict_label, accuracy , dec_values] = svmpredict(test_labels,test_data, model,'-b 1');
%}
```

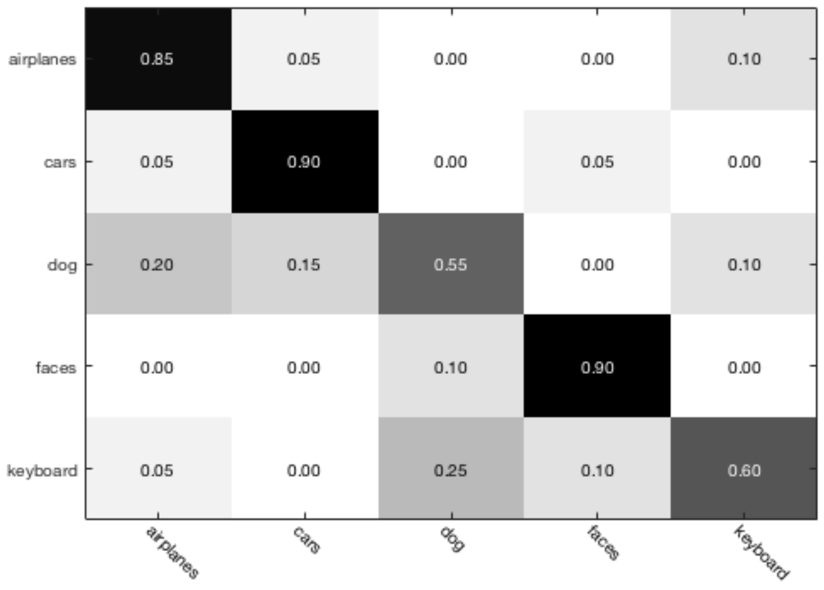
- Accuracy = 76% (76/100) (classification)

3. The overall and classification errors per class have been computed.

| Class | Error/Class |
|---|---|
| 1 | 0.15 |
| 2 | 0.10 |
| 3 | 0.45 |
| 4 | 0.10 |
| 5 | 0.40 |

| | |
|---|---|
| Mean overall error | 0.24 |

4. The confusion matrix has been computed and is shown below:



5. For each class, images that are correctly classified and images that are incorrectly classified are shown below:

**Only lab1.m and knnsearch.m have been submitted with the report as these were the files that have been modified. Otherwise it takes too much space and I am unable to upload the entire source files.**