Queen Mary University of London

Department of Electrical Engineering & Computer Science

ECS797P Machine Learning for Visual Data Analysis

Lab 1 – Image Classification using a Bag-of-Words Model

Hasan Emre Erdemoglu

4 March 2021

Table of Contents:

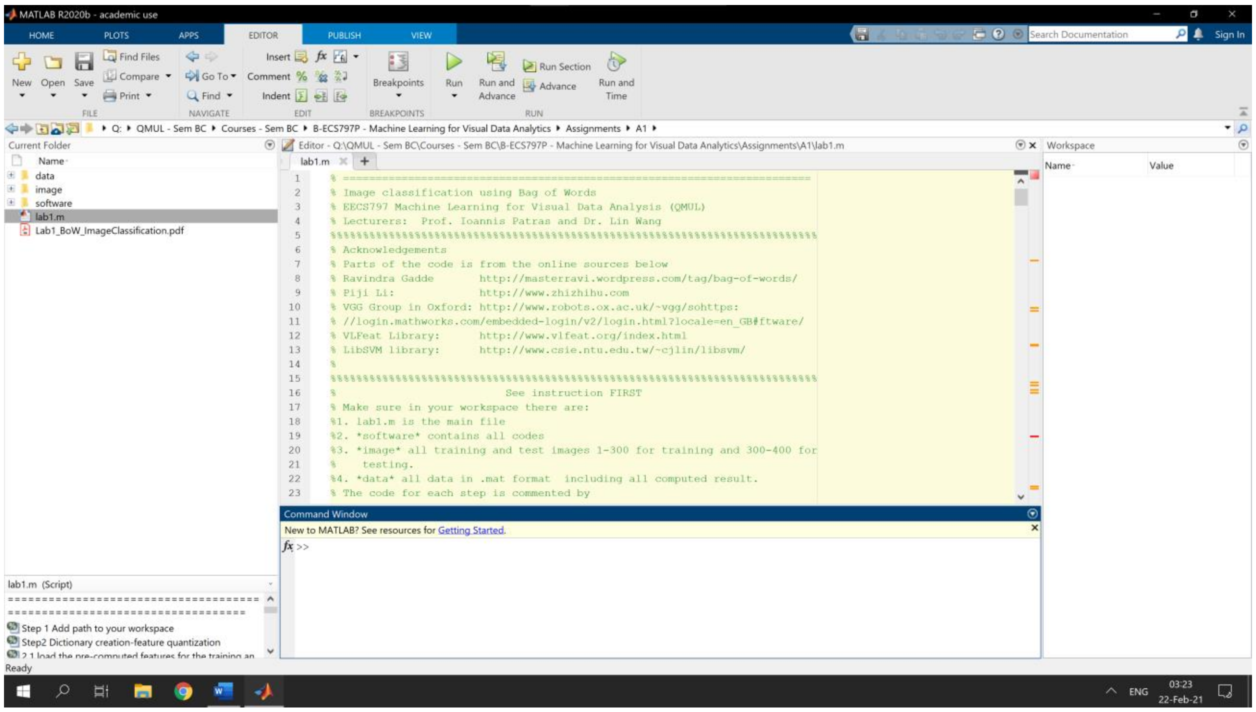
1. Getting Started

This laboratory report will examine Image Classification using Bag-of-Words (BoW), Nearest Neighbours (NN) representation and Support Vector Machine (SVM) classifiers. The implementations were done using MATLAB.

The root folder is named A1. This directory contains folders that are given with the assignment. The details of the contents of these folders are shown in the lab document.

When MATLAB is opened, the current folder pane is navigated to the root folder, A1. All directories and their subdirectories are included in the path. "software" path is explicitly added by MATLAB code supplied with the assignment, as it uses several libraries in this assignment.

For reference, the initial setup can be found below:



```
% ==========================================================================
% Image classification using Bag of Words
% EECS797 Machine Learning for Visual Data Analysis (QMUL)
% Lecturers:  Prof. Ioannis Patras and Dr. Lin Wang
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Acknowledgements
% Parts of the code is from the online sources below
% Ravindra Gadde       http://masterravi.wordpress.com/tag/bag-of-words/
% Piji Li:             http://www.zhizhihu.com
% VGG Group in Oxford: http://www.robots.ox.ac.uk/~vgg/sohttps:
% //login.mathworks.com/embedded-login/v2/login.html?locale=en_GB#ftware/
% VLFeat Library:      http://www.vlfeat.org/index.html
% LibSVM library:      http://www.csie.ntu.edu.tw/~cjlin/libsvm/
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          See instruction FIRST
% Make sure in your workspace there are:
%1. lab1.m is the main file
%2. *software* contains all codes
%3. *image* all training and test images 1-300 for training and 300-400 for
%    testing.
%4. *data* all data in .mat format  including all computed result.
% The code for each step is commented by
```

```
New to MATLAB? See resources for Getting Started.
>>
```

```
lab1.m (Script)
====================================
====================================
Step 1 Add path to your workspace
Step2 Dictionary creation-feature quantization
2.1 load the pre-computed features for the training an
```

Figure 1. Creating & Navigating to MATLAB Workspace for Assignment 1

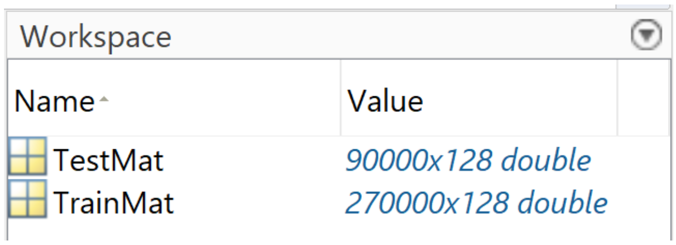
Note that some of the code and some steps are already provided with the report's lab documents. Code & detailed explanation of such sections will be omitted in this report, excluding the case where a description of these sections is vital for the questions asked in these or subsequent sections.

2. Dictionary Creation – Feature Quantisation

The specific parts of this question are divided into independent codes in the "lab1.m" file. This approach allows individual parts not to affect each other.

Part 1

The given code was used to import variables from the "all_features.mat" file to MATLAB's workspace. All dimensions are checked and matching with the lab document.


Workspace

Name	Value
TestMat	90000x128 double
TrainMat	270000x128 double

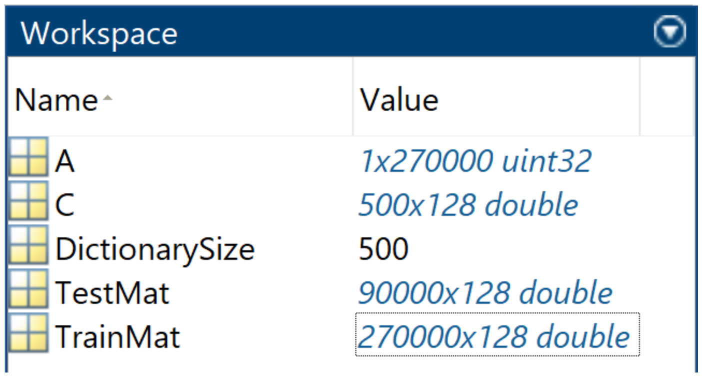
Figure 2. Loaded Variables

Part 2

The following MATLAB command needs to be used in the first run-time to run the assignment:

```
run('software/vlfeat-0.9.16/toolbox/vl_setup');% to compile the vlfeat lab.
```

After running this code "vlfeat" library will be available for the assignment. The code in this part uses "vl_kmeans" method to cluster a subset of extracted descriptors from the training dataset "TrainMat". It produces two variables, "A" (1x270000), which has the cluster assignment for each training sample and C (500x128), which grouped SIFT features for a dictionary size of 500 (controlled by parameter DictionarySize). C is saved by the code given in the assignment. Note that the SIFT feature has 128 dimensions. The following is the output:


Workspace

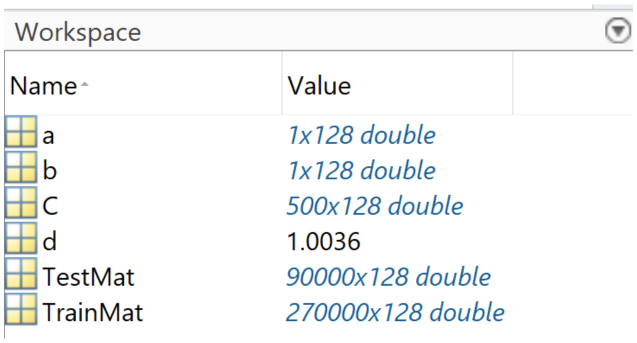
Name	Value
A	1x270000 uint32
C	500x128 double
DictionarySize	500
TestMat	90000x128 double
TrainMat	270000x128 double

Figure 3. Outputs A and C with sizes 1 by 270000 & 500 by 128, respectively.

Part 3

The code segment in Part 3 shows a demo of how "EuclideanDistance.m" should be used. It takes a single image descriptor from "TestMat" and uses a single codeword from "C" to calculate the distance between the descriptor and the codeword. Euclidean distance is calculated using the equation below. The figure below shows the output for this code segment. Each dimension is a SIFT feature, where we calculate the squared distance between the image descriptor and the codeword.

$$d(\bar{a},\bar{b}) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_{128} - b_{128})^2}$$


Workspace

Name	Value
a	1x128 double
b	1x128 double
C	500x128 double
d	1.0036
TestMat	90000x128 double
TrainMat	270000x128 double

Figure 4. Workspace output for Part 2.3.

In the figure above, "a" is a single sample from "TestMat" and "b" is a single cluster centre from the dictionary of cluster centres. "d" is the Euclidean distance between the given sample and the cluster centre. This part is already implemented within the lab document.

Part 4

The code written here repeats from Part 3. Each descriptor will be assigned to the closest cluster centre using the Euclidean distance. A sample implementation is also given with the laboratory assignment. The 3-line example code indicates the following: For each sample, Euclidean distance between the sample and the cluster centres can be calculated. "min()" operation can be used to map each sample to the cluster centre closest to it.

Only the written codes will be shown in this report. I have written the following code for this part:

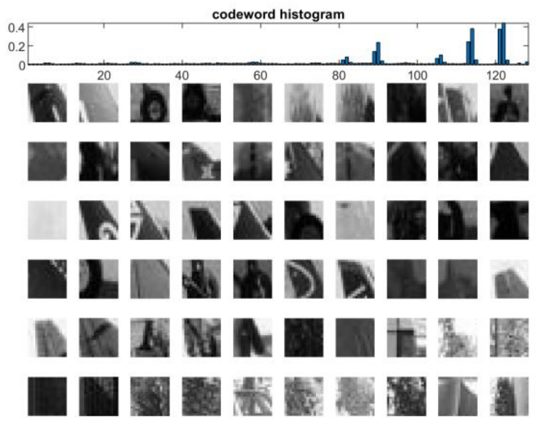
```
%-------Write Your Code here that assigns all descriptors -------------
% Using vectoral representation to save time.
[minTrain,index_train] = min(EuclideanDistance(TrainMat,C),[],2);
[minTest,index_test] = min(EuclideanDistance(TestMat,C),[],2);

% fixes notation wrt lab document we are given: (row vector representation)
index_train = index_train'; index_test = index_test';
save('data/global/assignd_discriptor','index_train','index_test');
%}
%-----------------------------end of code-----------------------------------
```

This implementation is analogous to the example given in this part. The only differences are it has been shrunk into a single line of code. Investigating EuclideanDistance.m", I have seen that this method is capable of performing matrix operations. Instead of looping over each sample, "TrainMat" and "TestMat" is given to the method. This operation is done to achieve vectorisation, to improve code efficiency.

EuclideanDistance(TrainMat, C) and EuclideanDistance(TestMat, C) will have sizes of 270000 by 500 and 90000 by 500, respectively. As minimum over the cluster centres is desired, the minimum is taken from the second dimension. Training and testing indices are transposed to comply with the lab assignment's notation on the dimensions. Finally, the training and testing indices are saved to memory.

Part 5 – (Not Available in Assignment Questions)

This part is not included in the laboratory document. It visualises some image patches assigned to the same codeword. "wordid" of 37, 78 and 397 are given as examples. The patch visualisations for the given inputs are listed below:



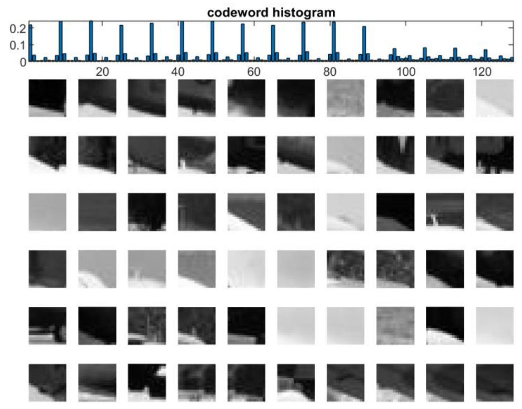


Figure 5. Output for words 37 (Left) and 397 (Right)

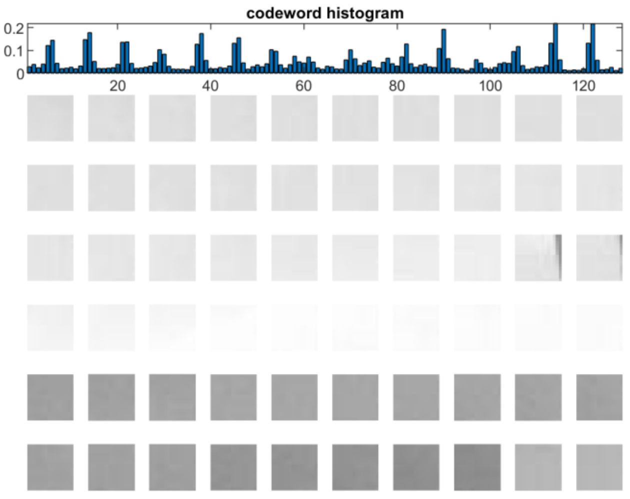


Figure 6. Output for word 78

3. Image Representation using Bag of Words Representation

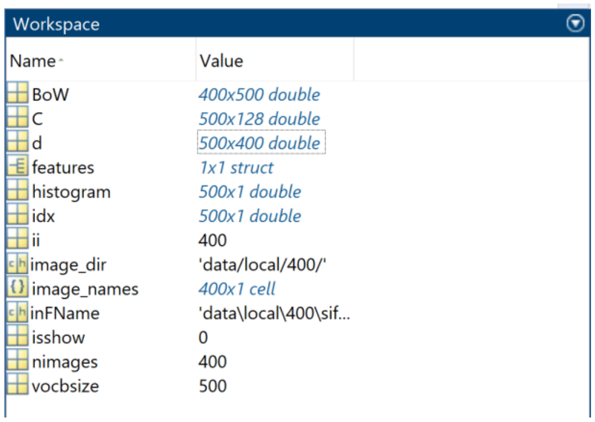
Part 1

In this part, the question requires representing training and test datasets as a histogram of visual words. The custom function "do_normalize.m" is used to normalise the histogram by L1 norm. The following code snippet achieves this operation:

```
      %------------ write your own code here-------------------------------
      d = EuclideanDistance(features.data, C); % Calc. dist.
      [~,idx] = min(d,[],2);
      [N,~] = histcounts(idx, 1:vocbsize+1);
      histogram = do_normalize(histogram);
      BoW(ii,:) = histogram; % save it to necessary variable

      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]);
        % pause; % for debugging
      end
end
save('data/global/bow','BoW')
%}
%------------------------end of code-----------------------------------
```

Note that the code snippet above is used within a for loop that iterates over 400 images. The first 300 images are reserved for training, and the remaining 100 images are reserved for testing. This code is equivalent to Part 2.4, where Bag of Words (BoW) representation is selected using an "idx" variable using min operation. Then the number of occurrences is counted per "idx" and normalised. The variable "idx" corresponds to the word's index for each of the descriptors (900 in total) within features.data. Also, note that dictionary size is set to 500 words throughout the assignment unless stated otherwise.

The following figure shows the variables in the workspace after running Part 3.1 from the "lab1.m" file:



Name	Value
BoW	400x500 double
C	500x128 double
d	500x400 double
features	1x1 struct
histogram	500x1 double
idx	500x1 double
ii	400
image_dir	'data/local/400/'
image_names	400x1 cell
inFName	'data\local\400\sif...
isshow	0
nimages	400
vocbsize	500

Figure 7. Workspace after running Part 3.1.

4. Image Classification using a Nearest Neighbour (NN) Classifier

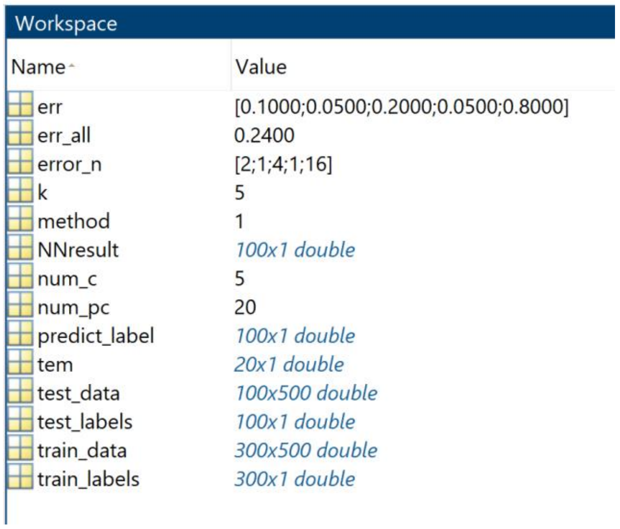
Part 1

Part 4 of this assignment will use the BoW variable generated by Part 3.1. The lab assignment already gives the code. This part uses a 1-NN search algorithm using L2 normalisation by default. "knnsearch.m" uses the arguments as provided in the lab assignment document where "T" is 100, "D" is 500 and "N" is 300. "k" is picked as 1.

The code in this part generates the labels for the images manually. Their label in the BoW variable is already ordered (the first 300 images are training images, images 301 to 400 is test images). Each class has 60 samples in the training dataset and 20 samples in the testing dataset. The method "knnsearch.m" is used to assign labels to the test data using the training data within the closest neighbour (the search is done between rows as defined by the method's documentation).

Part 2

The classification errors are computed in this part of the assignment. The lab assignment also gives the code for this section. The output of the workspace can be found below:


Workspace

Name	Value
err	[0.1000;0.0500;0.2000;0.0500;0.8000]
err_all	0.2400
error_n	[2;1;4;1;16]
k	5
method	1
NNresult	100x1 double
num_c	5
num_pc	20
predict_label	100x1 double
tem	20x1 double
test_data	100x500 double
test_labels	100x1 double
train_data	300x500 double
train_labels	300x1 double

Figure 8. Workspace after running Part 4.2.

Note that k in Part 4.1 and Part 4.2 are different variables. All parts work independently from each other, using saved variables from preceding code segments if necessary. From the workspace, it is seen that 24 % of the samples are classified incorrectly; more specifically, variable "error_n" describes how much samples are misclassified from the test dataset. In the outputs, class 1 has 18 correct classifications and two incorrect classifications. For the respective classes, the number of misclassifications is 1, 4, 1 and 16.

Part 3

The lab assignment also gives this part. The classes in Part 4.2 and 4.3 are airplanes, cars, dogs, faces, and keyboard, respectively. The following is the plot of the confusion matrix. The confusion matrix also shows how misclassifications per class are distributed to other classes and general information obtained through Part 4.2. For example, class 1 (airplanes) has only 1 mistake, where an airplane is mistaken with a keyboard (class 5)—the numbers from "error_n" and the outputs here match.

See the confusion matrix below:

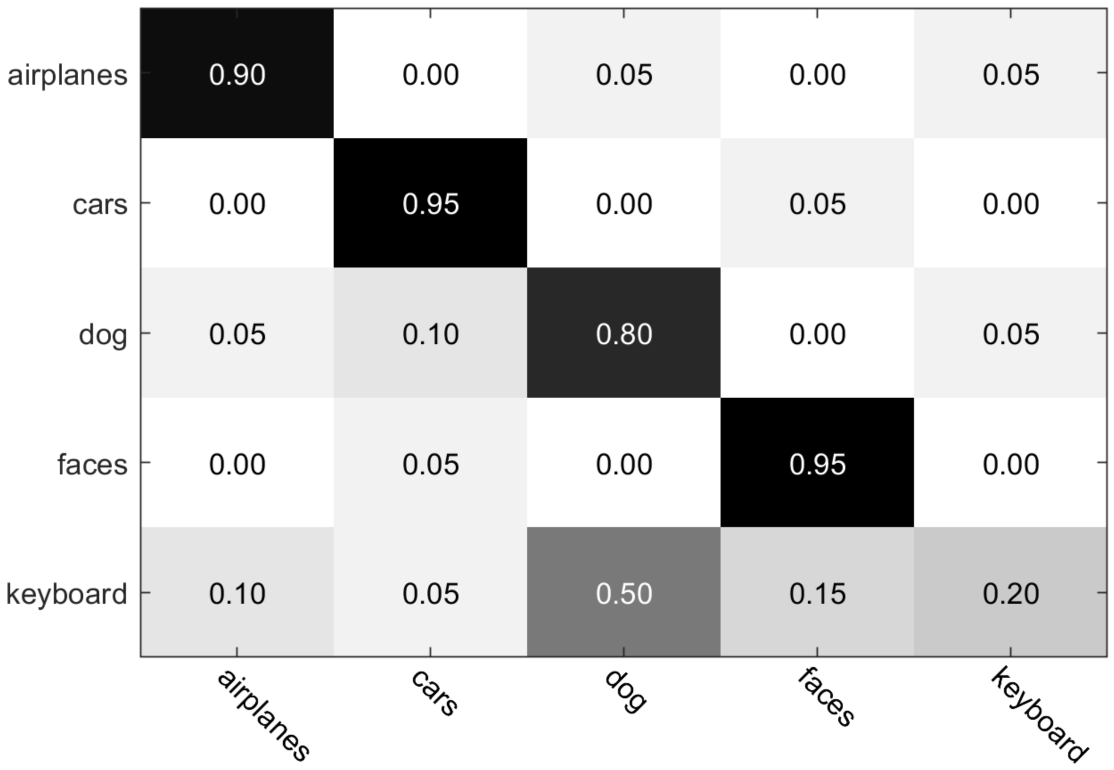


Figure 9. Confusion Matrix for Test Data using NN Classifier.

Part 4

This part of the code is already given in the assignment files. The following figures will show some correctly and incorrectly classified images along with some discussion about some of the failures:

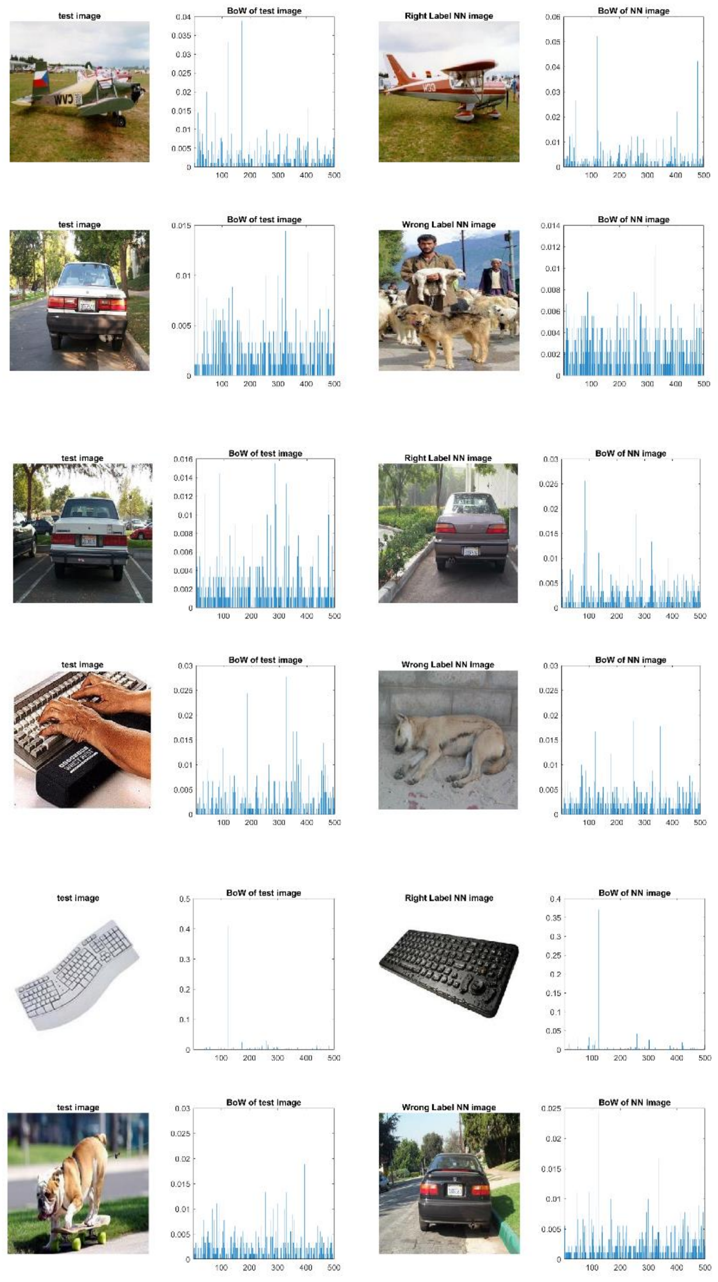


Figure 10. Some Correct Test Image and Label of the Image vs Incorrect Test Image and Label of the Image.

Figure 10 shows three correctly and incorrectly classified images. Test images are assigned classes using the training samples, using one nearest neighbour using Euclidean distances on Bag of Words representation. Test image will be assigned to the class of the training image where the words' overall distance is the closest without considering the distribution of the words. This would hinder the implementation's capability to look for specific words that may define an object class. Other words will smooth out the distance calculation, causing the algorithm to focus on all the words rather than keywords.

Notice that in all classifications (except for the correct keyboard case), the images have complex backgrounds, which give rise to multiple words being present all at once. This suppresses "key" words that define an image by making the BoW histogram more uniform. In dictionary space, this will lead to all classes clumping up together and using a distance metric that summarises the neighbour into a single numeric variable might make the classifier more prone to misclassification.

Part 5

Part 4.5 does the steps from 4.1 to 4.4 using the histogram intersection method for "knnsearch.m". I will implement this method. Histogram intersection was coded under "knnsearch.m". The implementation details and the code can be found below. Note that the code below assumes normalised histograms, which was already done in Part 3.1 when calculating BoW representation.

$$histint\left(h_i, h_j\right) = 1 - \sum_{m=1}^{K} \min\left(h_i(m), h_j(m)\right)$$

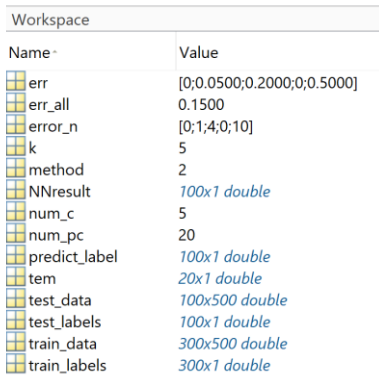
```
function d=histogram_intersection(a,b)

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

    end
```

The method is used as follows in “knnsearch” algorithm:

```
for k1 = 1:N
            for k2 = 1:L
                disk(k1,k2) = histogram_intersection(Q(k1,:),R(k2,:)); % one-to-one distance
            end
        end
```

Note that the intersection implementation loop over a and b, which are one by dictionary size (500) and the output distance is a one-by-one variable. "knnsearch" computes one-to-one bins using this implementation to construct the distance matrix, which is 100 by 300.

Running Parts 4.1 to 4.4 using histogram intersection method, the following overall and classification error per class is obtained:



Workspace	
Name	Value
err	[0;0.0500;0.2000;0;0.5000]
err_all	0.1500
error_n	[0;1;4;0;10]
k	5
method	2
NNresult	*100x1 double*
num_c	5
num_pc	20
predict_label	*100x1 double*
tem	*20x1 double*
test_data	*100x500 double*
test_labels	*100x1 double*
train_data	*300x500 double*
train_labels	*300x1 double*

Figure 11. Workspace after running Part 4.2 with Histogram Intersection

Figure 11 indicates the 15 % of the classifications were incorrect, where class 1 (airplanes) and 4 (faces) were identified all the time correctly. Class 2 (Car) was confused for one time only. Class 3 (Dogs) were confused four times, and Class 5 (keyboards) were confused ten times. Note that all categories have 20 test samples in the test set.

For histogram intersection, the following confusion matrix was produced:

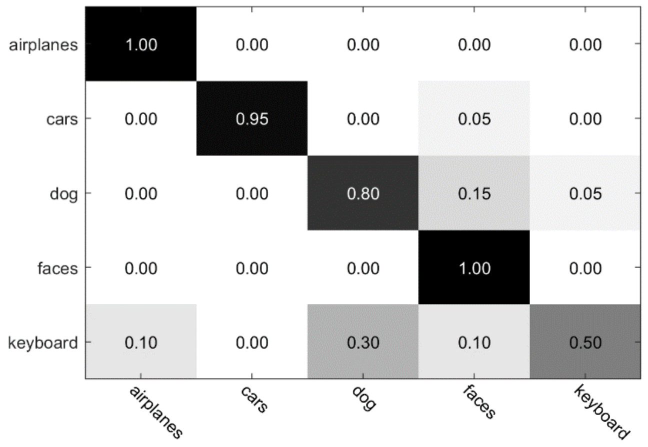


Figure 12. Confusion Matrix for Test Data using NN Classifier with Histogram Intersection

The confusion matrix gives more information on how the misclassifications are distributed in the class space. Cars had one misclassification, in which the car was classified as a face. Similarly, dogs and faces are misclassified, which is probable given dog faces, and human faces have some similar properties. Keyboards are misclassified 50 % of the time; the error is almost evenly distributed among all other classes. The following correct and incorrect classifications were obtained; three samples are given for correct and incorrect classifications:

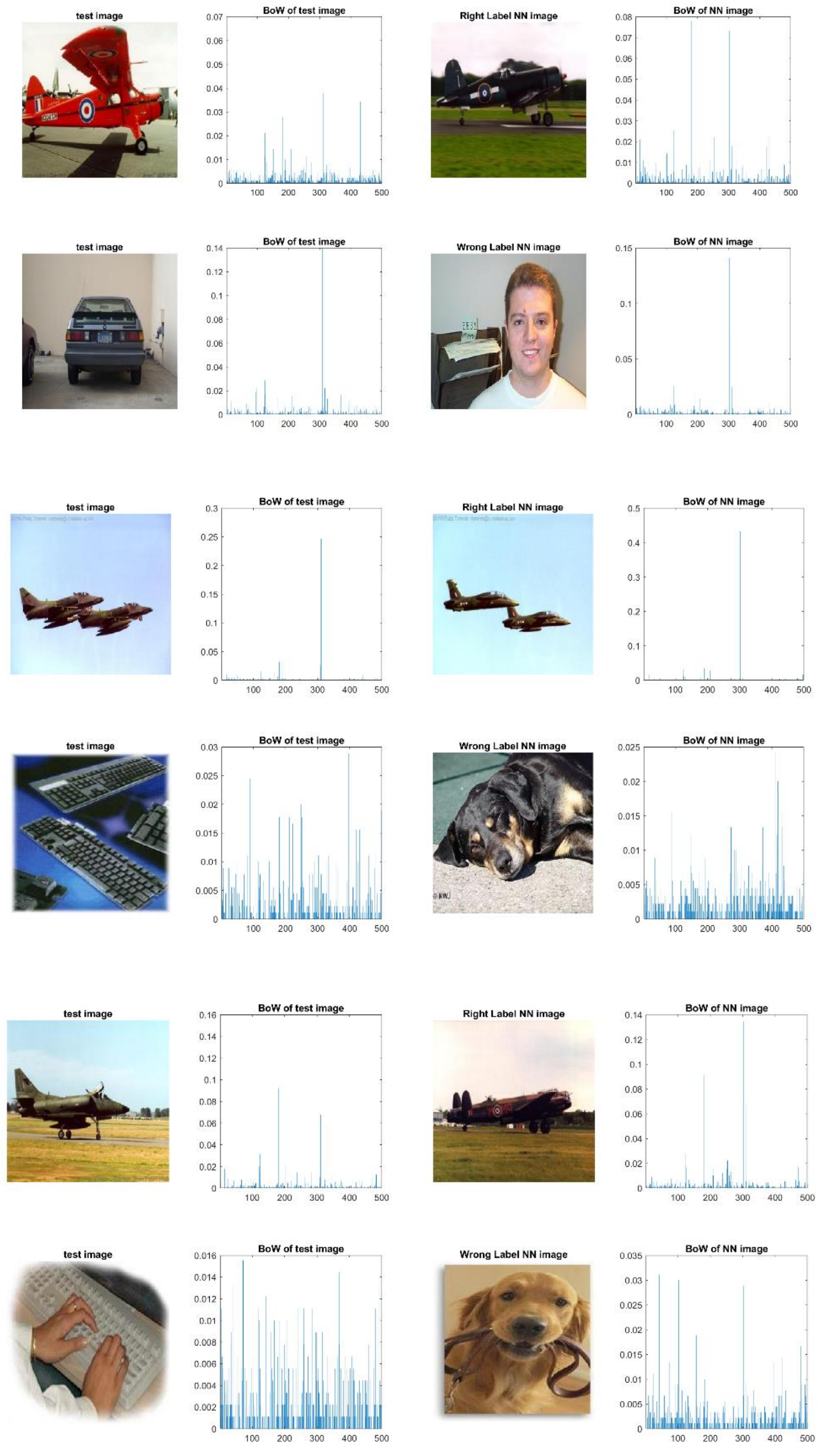


Figure 13. Some Correct Test Image and Label of the Image vs Incorrect Test Image and Label of the Image (Using Histogram Intersection Method).

Histogram intersection proved to be a better method than the L2 norm by reducing the overall misclassification rate from 24 per cent to 15 per cent. This could be because histogram intersection compares not only the distance of a test image BoW representation to all training BoW representation using a single distance metric; it compares each bin's contribution individually. Therefore, the overall shape and intersection between the training and test representations are also utilised in determining the class of the testing data.

But unlike the L2 method, the histograms for the correct classifications are sparser, implying that the classifier picked more matching histograms rather than optimising over L2 distance overall visual words present in the sample. The performance increase indicates that the histogram intersection also considers the overall shape of the visual-words distribution.

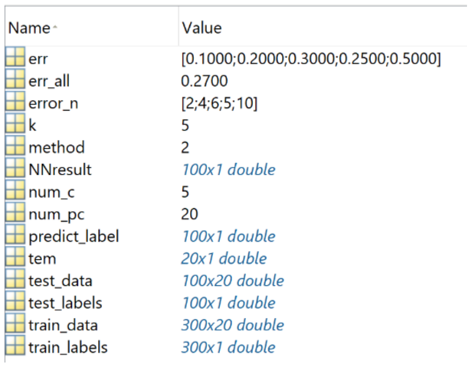
5. Dictionary Size

This experiment will be done using Euclidean and histogram intersection methods to assess small dictionary size performance in more depth. Explanation of the methods is omitted as it follows from the previous sections. As the implementation details are given in the section above, this section will only present the results. This section will use a dictionary size of 20.

Part 1

Using L2 Method

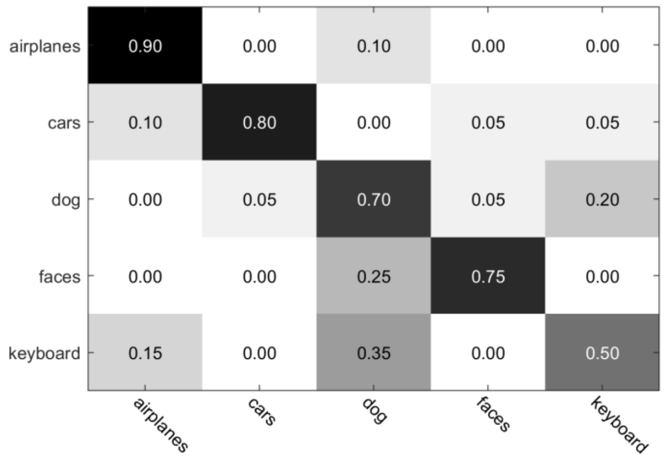
The following workspace shows the classification errors. Overall classification error increases to 27% with 2,4,6,5,10 misclassifications out from 20 samples, respectively.



Name	Value
err	[0.1000;0.2000;0.3000;0.2500;0.5000]
err_all	0.2700
error_n	[2;4;6;5;10]
k	5
method	2
NNresult	*100x1 double*
num_c	5
num_pc	20
predict_label	*100x1 double*
tem	*20x1 double*
test_data	*100x20 double*
test_labels	*100x1 double*
train_data	*300x20 double*
train_labels	*300x1 double*

Figure 14. Overall Classification and Class Classification Errors Using a dictionary size of 20 and L2 method

The confusion matrix is as follows:

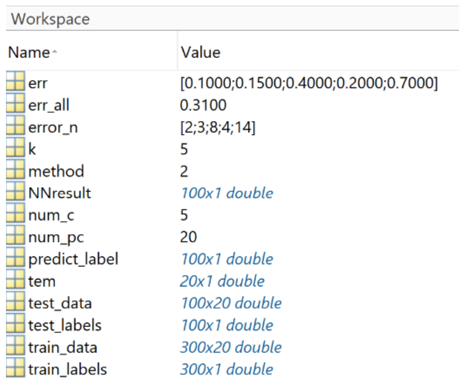


	airplanes	cars	dog	faces	keyboard
airplanes	0.90	0.00	0.10	0.00	0.00
cars	0.10	0.80	0.00	0.05	0.05
dog	0.00	0.05	0.70	0.05	0.20
faces	0.00	0.00	0.25	0.75	0.00
keyboard	0.15	0.00	0.35	0.00	0.50

Figure 15. Confusion matrix using a dictionary size of 20 and using L2 method.

Using the Histogram Intersection Method

The following workspace shows the classification errors. Overall classification error increases to 31% with 2,3,8,4,14 misclassifications out from 20 samples, respectively.

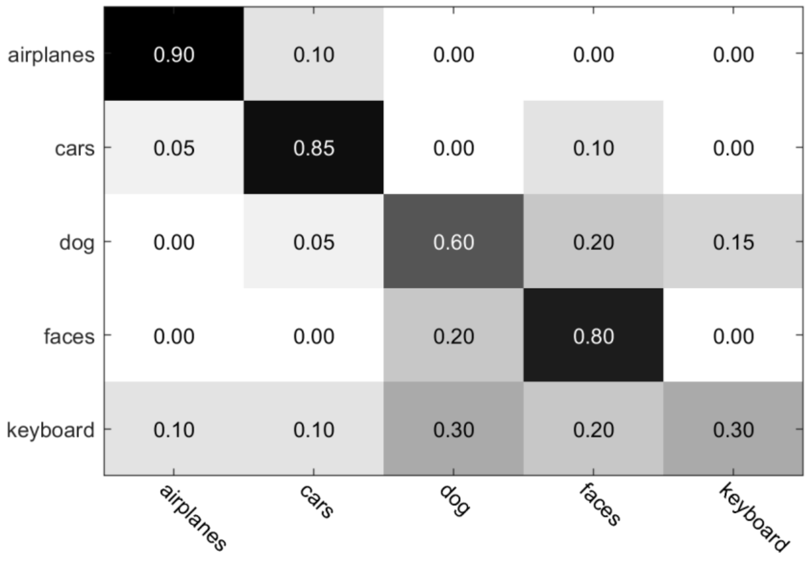


Workspace

Name	Value
err	[0.1000;0.1500;0.4000;0.2000;0.7000]
err_all	0.3100
error_n	[2;3;8;4;14]
k	5
method	2
NNresult	100x1 double
num_c	5
num_pc	20
predict_label	100x1 double
tem	20x1 double
test_data	100x20 double
test_labels	100x1 double
train_data	300x20 double
train_labels	300x1 double

Figure 16. Overall Classification and Class Classification Errors Using a dictionary size of 20 and Histogram Intersection method

The confusion matrix is as follows:



	airplanes	cars	dog	faces	keyboard
airplanes	0.90	0.10	0.00	0.00	0.00
cars	0.05	0.85	0.00	0.10	0.00
dog	0.00	0.05	0.60	0.20	0.15
faces	0.00	0.00	0.20	0.80	0.00
keyboard	0.10	0.10	0.30	0.20	0.30

Figure 17. Confusion matrix using a dictionary size of 20 and using the Histogram Intersection method.

Part 2

As the number of words is reduced to 20, "wordid" in part 2.4 is changed to 15 to show an example of visual word descriptors.

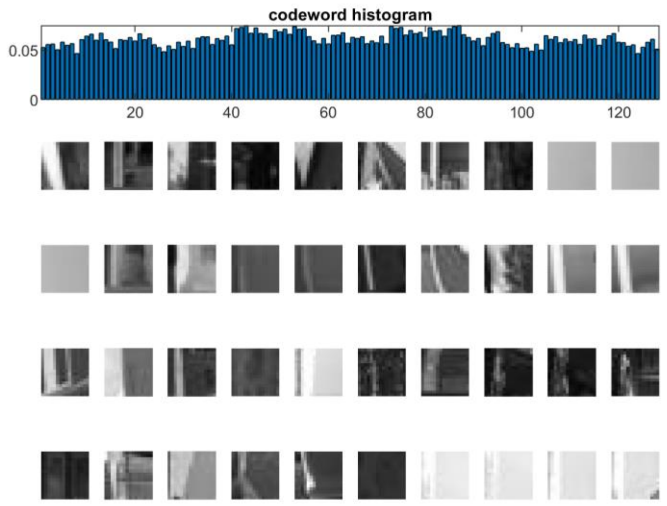


Figure 18. A sample of visual patches assigned to the same codeword (Using dictionary Size of 20)

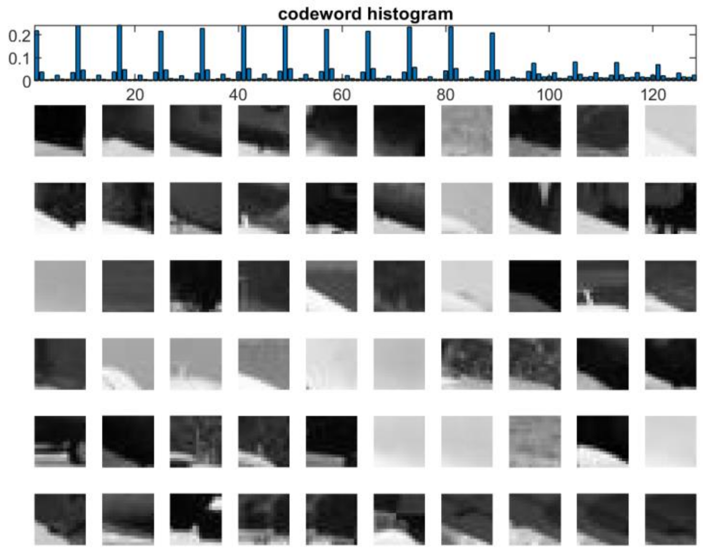


Figure 19. A sample of visual patches assigned to the same codeword (Using dictionary Size of 500)

In both L2 and Histogram Intersection methods, when a dictionary size of 20 is used, the classifiers' performance is dropped significantly. This is because when the number of codewords is small, image patches are summarised to a smaller number of codewords, which means visual words are not representative of the patches grouped under them.

Comparing Figure 18 with 19 shows that when 20 codewords are used, many different types of image patches are clustered to the same codeword. When a test image is being classified, as the number of codewords small, different patches are classified into the same codeword, which may cause the algorithm to misclassify the image. In Figure 19, this is not the case, as very similar patches are clustered within a codeword.

Also, notice that the SIFT histogram of the codeword in Figure 18 resembles a uniform distribution (high entropy, low information). This means within this codeword; all SIFT features are almost uniformly found. From an information theory perspective, it means that the codeword is not selective for any SIFT feature. In Figure 19, the codeword is selective for certain SIFT features as they are more frequently seen within the codeword.

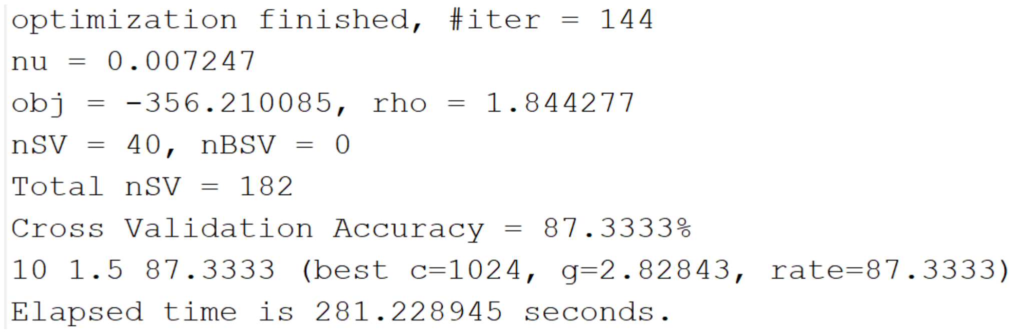
This means that using a dictionary size of 500 clusters image patches better than the codewords. Having a high amount of codewords all algorithm to utilise better the information contained in the patches increasing the overall accuracy of the classification. However, if the dictionary size were larger, each patch would represent its codeword (in the extreme case); that may cause overfitting and quantisation artefacts.

6. Image Classification using a Support Vector Machines (SVM) Classifier

This section will use an SVM classifier to predict labels for the test images using the BoW representations. It will use the "libsvm" library for the calculations. The documentation[1] of libsvm indicates that C is the cost and g is the gamma parameters for the SVM classifier.

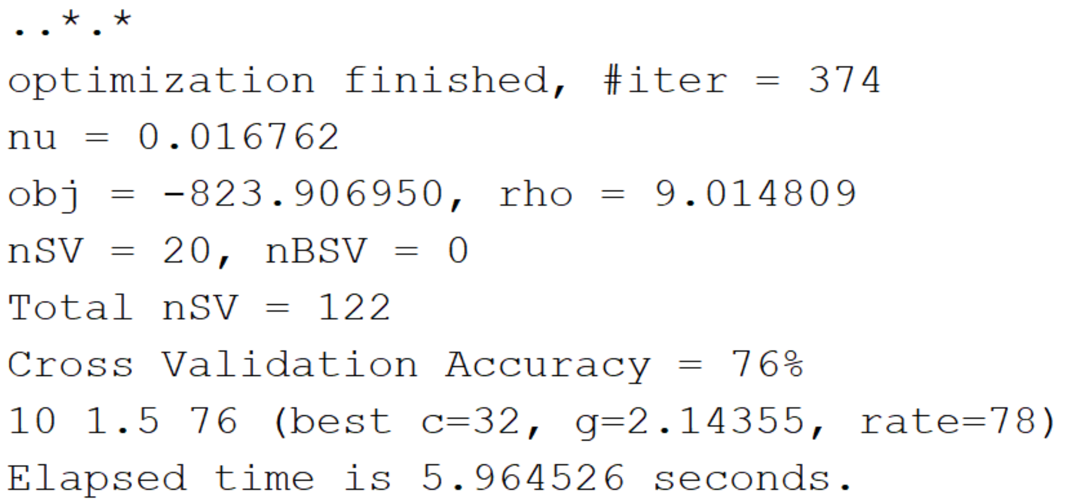
Part 1

The lab assignment gives the code for this part. The code runs multiple times to determine the best parameters for the SVM classifier. The cross-validation is done by systematically running a grid search over parameters C and g.



```
optimization finished, #iter = 144
nu = 0.007247
obj = -356.210085, rho = 1.844277
nSV = 40, nBSV = 0
Total nSV = 182
Cross Validation Accuracy = 87.3333%
10 1.5 87.3333 (best c=1024, g=2.82843, rate=87.3333)
Elapsed time is 281.228945 seconds.
```

Figure 20. CV Output of the SVM Classifier. Best C and g is selected as 1024 and 2.83, respectively. (Using dictionary size of 500)



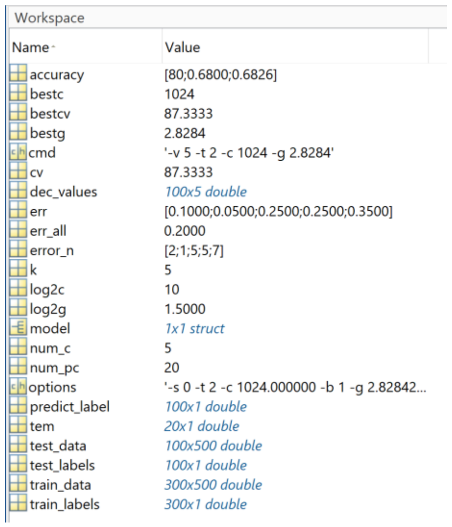
```
..*.*
optimization finished, #iter = 374
nu = 0.016762
obj = -823.906950, rho = 9.014809
nSV = 20, nBSV = 0
Total nSV = 122
Cross Validation Accuracy = 76%
10 1.5 76 (best c=32, g=2.14355, rate=78)
Elapsed time is 5.964526 seconds.
```

Figure 21. CV Output of the SVM Classifier. Best C and g is selected as 32 and 2.14, respectively. (Using dictionary size of 20)

[1] Reference on: https://www.csie.ntu.edu.tw/~cjlin/libsvm/

Part 2 - 3

After applying the SVM on test images, the following classification errors were obtained. Overall classification accuracy is 80 %, with 2,1,5,5, and 7 misclassifications per each class, respectively. The details will be discussed in Part 6.4.

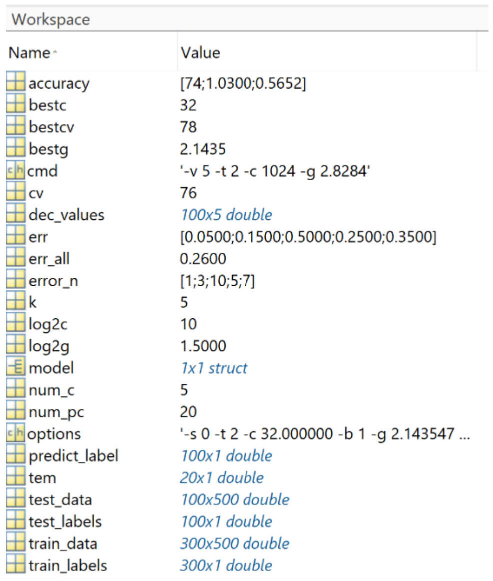


Workspace

Name	Value
accuracy	[80;0.6800;0.6826]
bestc	1024
bestcv	87.3333
bestg	2.8284
cmd	'-v 5 -t 2 -c 1024 -g 2.8284'
cv	87.3333
dec_values	100x5 double
err	[0.1000;0.0500;0.2500;0.2500;0.3500]
err_all	0.2000
error_n	[2;1;5;5;7]
k	5
log2c	10
log2g	1.5000
model	1x1 struct
num_c	5
num_pc	20
options	'-s 0 -t 2 -c 1024.000000 -b 1 -g 2.82842...
predict_label	100x1 double
tem	20x1 double
test_data	100x500 double
test_labels	100x1 double
train_data	300x500 double
train_labels	300x1 double

Figure 22. Classification Error Output (Part 2-3, Dictionary Size 500)

For a dictionary size of 20, the classification accuracy is 74 %. Each class is misclassified for 1, 3, 10, 5 and 7 times, respectively.



Workspace

Name	Value
accuracy	[74;1.0300;0.5652]
bestc	32
bestcv	78
bestg	2.1435
cmd	'-v 5 -t 2 -c 1024 -g 2.8284'
cv	76
dec_values	100x5 double
err	[0.0500;0.1500;0.5000;0.2500;0.3500]
err_all	0.2600
error_n	[1;3;10;5;7]
k	5
log2c	10
log2g	1.5000
model	1x1 struct
num_c	5
num_pc	20
options	'-s 0 -t 2 -c 32.000000 -b 1 -g 2.143547 ...
predict_label	100x1 double
tem	20x1 double
test_data	100x500 double
test_labels	100x1 double
train_data	300x500 double
train_labels	300x1 double

Figure 23. Classification Error Output (Part 2-3, Dictionary Size 20)

Part 4

In this section, KNN and SVM confusion matrices will be compared. See the following figure:

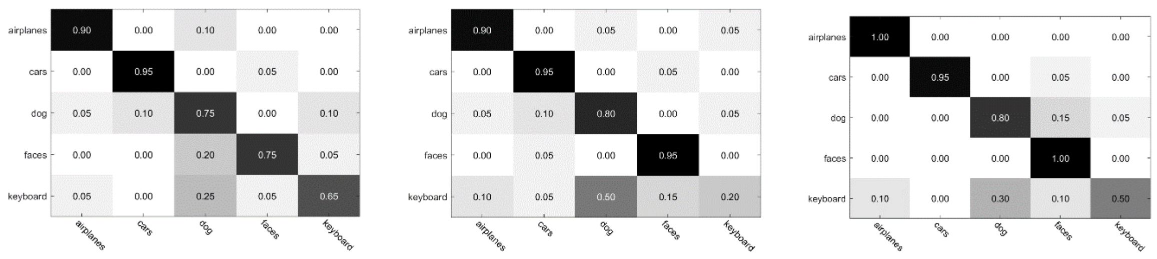


Figure 24. SVM Confusion Matrix (Left) vs KNN with L2 Method (Middle) vs KNN with Histogram Intersection Method (Right) for a dictionary size of 500.

SVM Classifier has an overall accuracy of 80%, where KNN has an overall accuracy of 76 % with L2 methods and 85% with the Histogram intersection method. While KNN with histogram intersection has the best overall classification, SVM can correctly classify keyboard class better than KNN with histogram intersection. KNN with the L2 method classifies the keyboards the worst even though overall performance for the other classes is higher. While having a worse classification rate per class, SVM is more balanced across the classes. This is mostly due to the soft margin it optimises when separating the decision boundary between classes.

The following figure does the same for a dictionary size of 20. A similar interpretation can also be stated for a dictionary size of 20. The only difference here is that the overall and class-level accuracy will be lower as 20 visual words cannot represent the image patches effectively.

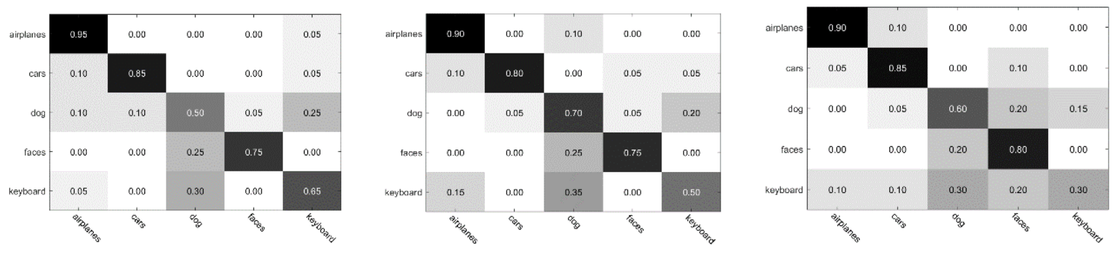


Figure 25. SVM Confusion Matrix (Left) vs KNN with L2 Method (Middle) vs KNN with Histogram Intersection Method (Right) for a dictionary size of 20.

Part 5

The following figure is the output of the SVM classifier using a dictionary size of 500:

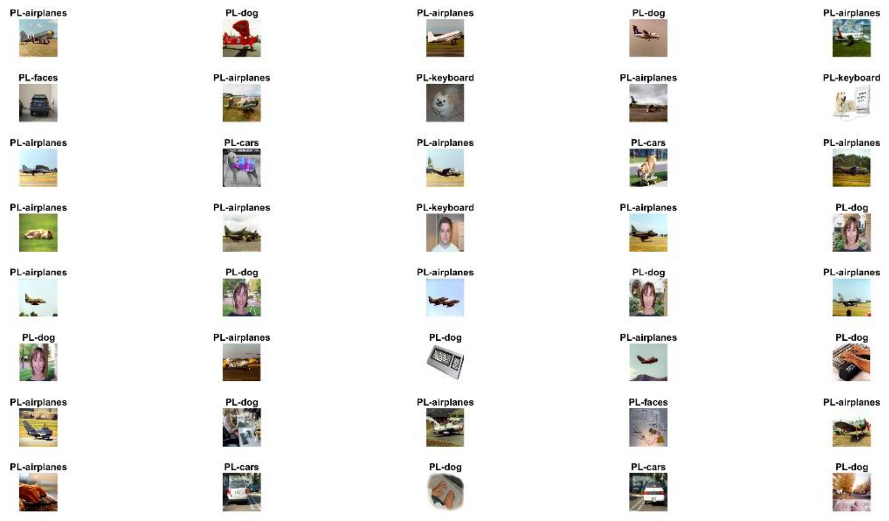


Figure 26. Correctly and Incorrectly Classified Images (Using a dictionary size of 500)

From figure 26, it can be said that dogs and keyboards are frequently mistaken. There are 5 sample images where dogs and keyboards are misclassified. The second most frequent misclassification is with dogs and faces, with 3 cases are happening within Figure 26. Mostly faces are described as dogs, but the opposite statement does not hold. In Figure 24, Left, this is also seen. Given the label is a dog, the image can be a face, but the opposite confusion did not happen. This is interesting as the confusions do not equally happen in both inter-class dimensions.

Figure 26 can be further investigated through the lens of the confusion matrix in Figure 24 for the other classes. It is visible that the confusion matrix results and example image-label pairs in Figure 26 match. A similar analysis and discussion can be done for a dictionary size of 20 as well. The image-label pairs using SVM for a dictionary size of 20 is given below for completeness purposes.

The following figure is the output of the SVM classifier using a dictionary size of 20:

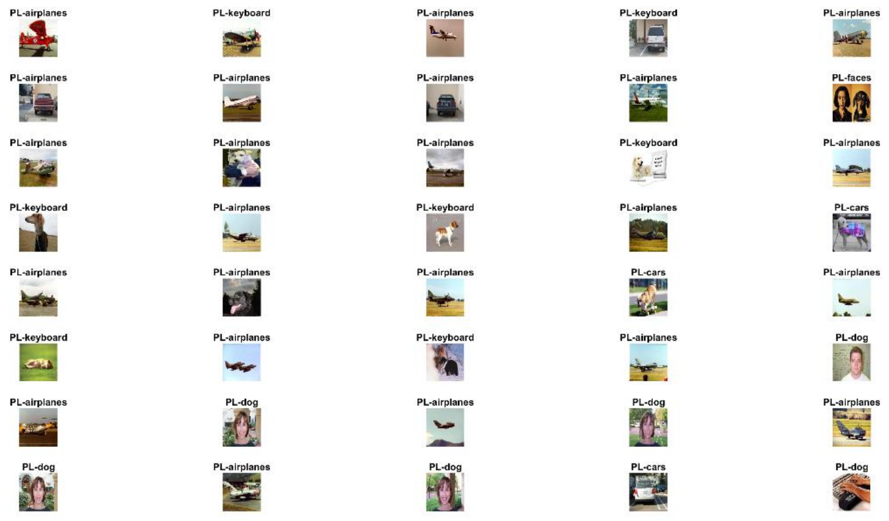


Figure 27. Correctly and Incorrectly Classified Images (Using a dictionary size of 500)