Machine Learining for Computer Vision Coursework 2
Visual Categorisation by BoW (K-means) and SVM

David Angelov
MEng Electrical and Electronic Engineering
Imperial College London
david.angelov12@imperial.ac.uk

Huaqi Qiu
MSc Communication and Signal Processing
Imperial College London
h.qiu15@imperial.ac.uk

1. Question 1

2. Question 2

2.1. Theory of SVM and multi-class extension

In this coursework, we applied Support Vector Machine (SVM) as the classifier. SVM is a binary classifier based on the discriminant function

$$y(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b \tag{1}$$

where $\boldsymbol{w}$ is a weight vector and b is the bias. The dataset x is classified to one of the two classes according to the value of $y(\boldsymbol{x})$. However, there are cases when dataset is not linearly separable, in which the Kernel Trick is required. The Kernel Trick map the data points to a higher-dimensional feature space so that the dataset is linearly separable. This process is denoted by: $\boldsymbol{x} \rightarrow \phi(\boldsymbol{x})$. Now the binary SVM function (Eq.1) becomes

$$y(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}}\phi(\boldsymbol{x}) + b \tag{2}$$

The SVM separate data in the Kernel transformed space by a linear decision hyperplane. The best hyperplane is found by maximizing the margin of separation. The margin is evaluated by finding the distance of the closest training point to the decision hyperplane. The optimised weighting vector $\boldsymbol{w}$ takes the form of

$$\boldsymbol{w} = \sum_{n=1}^{N} a_n t_n \phi(\boldsymbol{x_n}) \tag{3}$$

where $\boldsymbol{x_n}$ is training data vector and t_n is the labels of the n^{th} training data vector. Then the SVM decision function becomes

$$y(\boldsymbol{x}) = \sum_{n=1}^{N} a_n t_n k(\boldsymbol{x}, \boldsymbol{x}_n) + b \tag{4}$$

where $k(\boldsymbol{x}, \boldsymbol{x}_n)$ is the Kernel function. In this course work we primarily consider the RBF kernel function, which is formulated as

$$k(\boldsymbol{x}, \boldsymbol{x'}) = exp(\frac{||\boldsymbol{x} - \boldsymbol{x'}||^2}{2\sigma^2}) \tag{5}$$

We considered the parameter σ in the kernel function as the primary factor that affects the performance of RBF kernel SVM. We also considered the parameter C, which is an error weighting constant that penalizes the misclassification during the supervised learning and effectively controls the margin.

Since SVM is two-class classifier, we extended it in order to solve multi-class classification problems. Two methods of extension were considered:

1. One versus the rest (OVR)
   In OVR method, we trained M separate SVMs for M class problem. In the training of the m^{th} SVM, the m^{th} class were labelled as positive class, while the rest $M-1$ classes were labelled as negative class. Upon testing a query image $\boldsymbol{x}$, the output from all M SVMs were compared. The query image was classified as the m^{th} image if the output of the m^{th} SVM was the highest, i.e.
   $$y(\boldsymbol{x}) = max_m\{y_m(\boldsymbol{x})\} \tag{6}$$
2. One versus one (OVO)
   In OVO method, we trained SVMs by pairs of the classes. Each of the $M(M-1)/2$ SVMs was trained by data from a pair of two classes, labelling data points from one of the class as the positive class. In testing, the query image was tested and the output from each SVM was treated as 'vote'. The class that had the majority of votes was the class that the query image was classified to.

2.2. Implementation on Toy Spiral data

In this coursework, we firstly tested the SVM theory on the Toy Spiral data, shown in Figure 1. Each data point is in

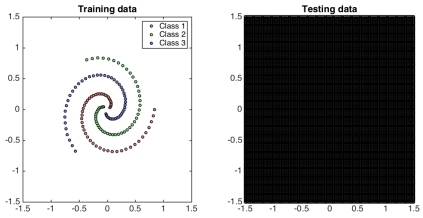


Figure 1: Toy Spiral training and testing data face

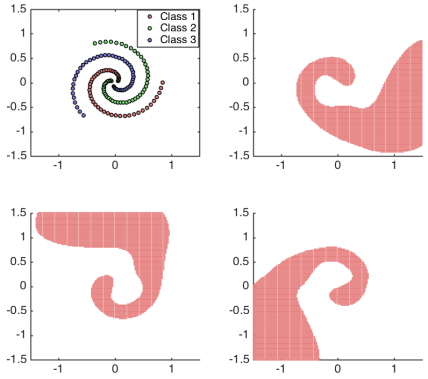


Figure 2: Result form each OVR extended SVMs

the form of $[(X_n, Y_n), t_n]$, where (X_n, Y_n) is 2-D Cartesian coordinates and t_n is the class label of the data point. The testing data is a 2D dense grid within the range of $X_n, Y_n \in [-1.5, 1.5]$. Testing with this testing data set can visualise the decision boundaries.

2.3. SVM extension

As introduced previously, we extended the binary class SVMs to multi-class SVMs by OVR and OVO. Figure 2 shows the classification results from each of the 3 SVMs extended by OVR method. In this Figure, red regions represents the positive class classified by the SVM. The combined classification of the 3 SVMs is shown in Figure 3. In all demonstrations in this part of the discussion, the kernel coefficients σ and C are kept unchanged ($\sigma = 0.8$ and $C = \infty$).

Similarly, Figure 4 and 5 show the results of individual SVMs and their combination respectively. Notice that in the OVO case, the votes from all $M(M-1)$ might not have a single majority. Data points with these classification results were classified as 0 in this question, which corresponds to the 'white' area in Figure 5. For the recognition problem of Caltech 101 data in question 4, we randomly selected class labels from all classes for these type of data points by `randi`.

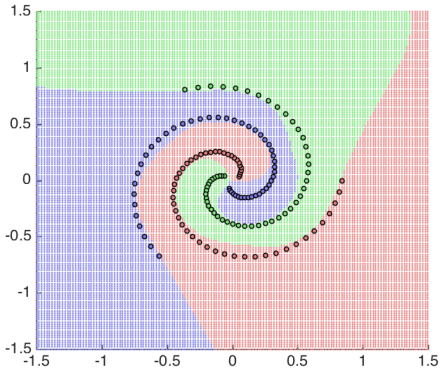


Figure 3: Combined result of OVR extended SVMs

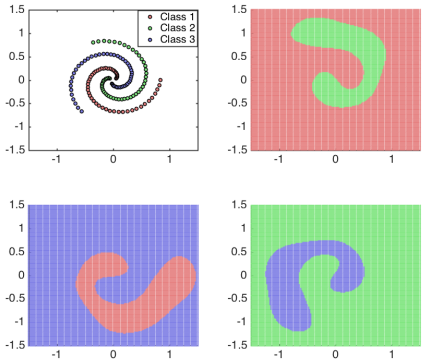


Figure 4: Result form each OVO extended SVMs

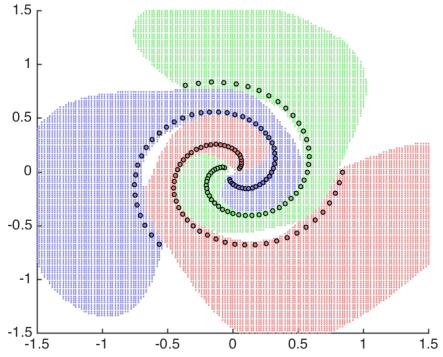
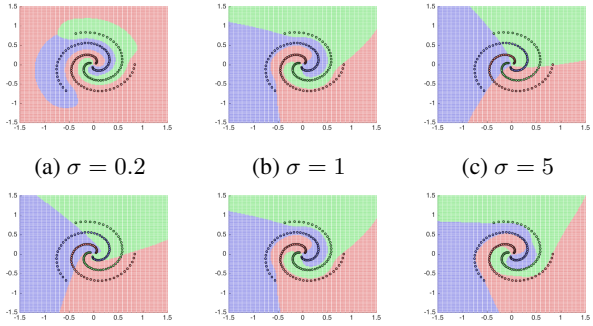


Figure 5: Combined result of OVO extended SVMs

2.4. SVM parameters

The parameters σ for the RBF kernel function and the parameter C for the misclassification penalty are problem



(a) $\sigma = 0.2$ (b) $\sigma = 1$ (c) $\sigma = 5$

(d) $C = 1$ (e) $C = 1000$ (f) $C = \infty$

Figure 6: Classification results with varying parameters σ and C for OVR

dependent. Here we discuss the effect of the two parameters using the Toy Spiral data test. The kernel parameter σ affects the generalization or complexity of the model. Simpler models could have better generalization to unseen data with the risk of underfitting. Complex models on the other hand could be overfitted to training data and results in high testing loss (i.e. lower classification accuracy). This can be reflected on the fact that the more complex the system gets, the more support vectors are utilised to construct the marginal boundaries. Figure (a), (b) and (c) in Figure 6 show the result of varying the parameter σ while keeping C unchanged. Result (a) demonstrates the case of overfitting and (c) demonstrates the case of underfitting.

The parameter C for each misclassified data point is proportional to its distance from the decision boundary. This indicates that C controls the margin allowed for the SVMs. The plots (d), (e) and (f) in FIgure 6 present the classification results of varying the parameter C while keeping σ unchanged. The result of (d) suggests that the model was underfitting when C is low, while (e) and (f) indicate that the performance of the model could converges when C is sufficiently large. We will further discuss the effects of the parameters σ and C in a numerical manner (accuracy) for the Caltech101 data set.

3. Question 3

In this question, we performed image recognition using the Caltech101 data set. We randomly selected 15 images from each of the 10 different classes. The performance of the classification was evaluated by the correction rate of image classification. We investigated the following parameters that could affect the accuracy:

- Vocabulary size = K
- Kernel types (RBF and Polynomial) and kernel parameters (σ for RBF, polynomial order M for polynomial)
- Misclassification penalty parameter C

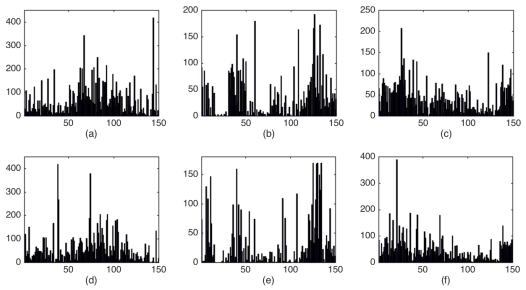


Figure 7: Histogram of sample training images: (a), (b) & (c) training images, (d), (e) & (f)

3.1. Bag of Words

Both training and testing images are transformed into data points for classification by the Bag-of-Words technique, which utilized the K-means clustering in Question 1. This process can be described as:

1. Visual vocabulary construction (using K-means clustering)
   We firstly apply the Scale Invariant Feature Transform (SIFT) descriptor to extract features (visual words) from each image. Each SIFT descriptor $\boldsymbol{d}$ has a dimension of $D = 128$. Then, we selected 100,000 descriptors and applied the K-means clustering. The K cluster centres found are called codewords or visual vocabulary.

2. Histogram of visual words
   Each image was represented by a histogram of visual words, which is constructed by assigning the visual words to the codewords by the Nearest Neighbour matching and counting the number of words assigned to each codeword. The distance between visual words and codewords can be evaluated by the Euclidean distance, i.e.

$$E(\boldsymbol{d_1}, \boldsymbol{d_2}) = ||\boldsymbol{d_1} - \boldsymbol{d_2}|| \tag{7}$$

Figure 7 shows the histogram of some example training and testing images. The recognition problem was transformed to a data point based classification problem, which was then solved by the multi-class SVM classification. The accuracy of recognition was affected by value of K as shown in Figure 8. The correction rate increased rapidly when the increasing K was small. However, we also observed that the accuracy converged to maximum rate as K

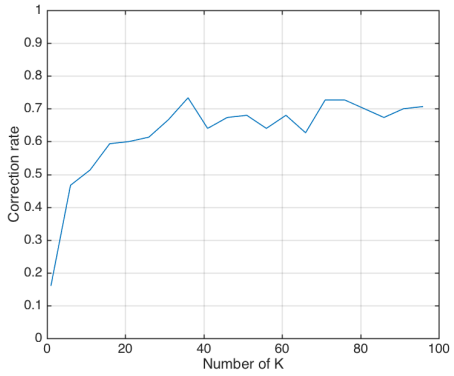


Figure 8: Accuracy affected by the number of K

reached around 50. The complexity of the SVM is dependent on the number of support vectors and the data dimension (K). Hence, it's optimum to set M at 50 to 80 to avoid overcomplexity.

3.2. OVO and OVR

The method of multi-class extension of binary SVMs could affect the accuracy of classification. To investigate this, we fixed the other parameters as: RBF kernel, $\sigma = 2^{15}$, $C = 2^{15}$ and $K = 50$, and tested both OVO and OVR extended SVMs. The correction rates were averaged over 10 iterations to reduce the variance. The resulting accuracy rates are shown in Table 1, which states that the OVR method averagely has 2 % accuracy more than OVO method.

Table 1: Comparing OVO and OVR

	OVR	OVO
Accuracy	66.67%	64.27 %

3.3. Kernel type

In this coursework, we applied and compared two types of kernel functions: RBF function (Eq.5) and Polynomial kernel function

$$k(\boldsymbol{x}, \boldsymbol{x'}) = (\boldsymbol{x^T x'} + c)^M \tag{8}$$

where M is the polynomial order. We trained the SVMs with two kernel functions separately and compared their performances. The RBF parameters σ was set to `'auto'`, which enables the MATLAB function `fitcsvm` to automatically select an optimum value. The Polynomial order M was set to 3. OVR extension was used for both kernel

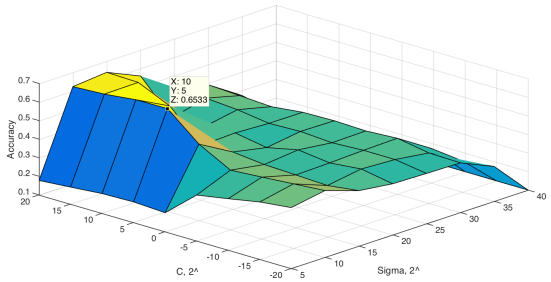


Figure 9: Accuracy varying both σ and C

functions. The accuracy of such configuration is shown in Table.2. The results can only suggest that the polynomial kernel function with this specific configuration yields better accuracy. However, the results were subjective to the parameters and more importantly, the data sets used. The two kernel functions generates different high dimensional kernel spaces, which should be optimum for different types of data sets.

Table 2: Comparing RBF and Polynomial kernel function

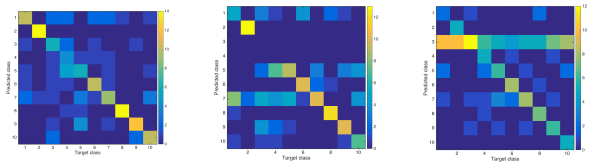
	RBF	Polynomial
Accuracy	63.82%	65.00 %

3.4. Kernel parameter σ and C

Recall that in Question 2, the kernel parameter for RBF (σ) affected the overfitting and underfitting of the model, which affects the performance of the model with new testing data points. The parameter C which controls the margin of each SVM also affect the generalization of the trained models. We trained and tested OVO extended SVMs and evaluated the correction rate varying the σ from 2^{-5} to 2^{40} while varying the parameter C from 2^{-20} to 2^{20}. The result is visualized in Figure 9. The accuracy for each parameter pair $\{\sigma$,C$\}$ was averaged over 10 iterations. Figure 9 can be summarized as:

- Increasing C increased the accuracy when C is below 2^5. Further increasing C did not increase the accuracy rate. However, higher value of C may lead to overfitting if the model is to be tested by new testing data since it lead to smaller margin;
- The accuracy has a peak with respect to σ and reduces when σ was large. This verified the theory that σ could lead to overfitting and increasing of testing lost.

Figure 10 contains the confusion matrices for some combinations of $\{\sigma, C\}$



(a) $\sigma = 2^{10}$, $C = 2^5$

(b) $\sigma = 2^{10}$, $C = 2^0$

(c) $\sigma = 2^7$, $C = 2^5$

Figure 10: Confusion matrices

References

[1] A. Alpher. Frobnication. *Journal of Foo*, 12(1):234–778, 2002.

[2] A. Alpher and J. P. N. Fotheringham-Smythe. Frobnication revisited. *Journal of Foo*, 13(1):234–778, 2003.

[3] A. Alpher, J. P. N. Fotheringham-Smythe, and G. Gamow. Can a machine frobnicate? *Journal of Foo*, 14(1):234–778, 2004.

[4] Authors. The frobnicatable foo filter, 2014. Face and Gesture submission ID 324. Supplied as additional material `fg324.pdf`.

[5] Authors. Frobnication tutorial, 2014. Supplied as additional material `tr.pdf`.

A. Appendix 1 MATLAB Code

A.1. One vs Rest extension

```
% M-SVM 1 vs Rest function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input
% data_train = [dim_train x (K+1)], last column is label
% data_test = [dim_test x (K+1)]
% M = number of classes
% kernel = kerner function type

% Output
% predict_label = [dim_test x 1]
%                 predicted label for each row of testing data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function predict_label = fMSVM_1vR(data_train, data_test, kernel, C, sigma)

score = [];
M = length(unique(data_train(:,end)));
% num_supp_vectors = [];

for class = 1:M

% Re-organise label for 1 vs Rest
data_class = ones(numel(data_train(:,end)),1);
data_class(data_train(:,end) ~= class) = -1;

    switch kernel
    case 'RBF'
    SVM = fitcsvm(data_train(:,1:end-1), data_class,...
                'KernelFunction', 'rbf',...
                'BoxConstraint',C,...
                'KernelScale', sigma,...
                'ClassNames',[-1,1]);

    case 'Poly'
    SVM = fitcsvm(data_train(:,1:end-1), data_class,...
                'KernelFunction', 'polynomial',...
                'BoxConstraint', 1, ...
                'KernelScale', sigma, ...
                'PolynomialOrder',C, ...
                'ClassNames',[-1,1]);
    end

%   num_supp_vectors = [num_supp_vectors, length(SVM.SupportVectors)];

% test SVM
[~, score_one] = predict(SVM, data_test(:,1:(end-1)));
score_one_norm = score_one(:,2);

% score_one_norm = (score_one_norm - min(score_one_norm))/(max(score_one_norm) - min(score_one
```

```
score_one_norm = 1./(1 + exp(-score_one_norm));

score = [score, score_one_norm];

end

[~, predict_label] = max(score,[],2);

display('1 vs Rest SVM training and testing completed');

end
```

A.2. One vs one extension

```
% M-SVM 1 vs 1 function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Input
% data_train = [dim_train x (K+1)], last column is label
% data_test = [dim_test x (K+1)]
% M = number of classes

% Output
% predict_label = [dim_test x 1]
%                 predicted label for each row of testing data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function predict_label = fMSVM_1v1(data_train, data_test, kernel, C, sigma)

data_class = data_train(:,end);

class_type = unique(data_train(:,end));
M = length(class_type);
%num_svm = M*(M-1)/2;
vote = [];
Kernel_Scale = sigma;

for id_svm_i = 1 : M - 1
    for id_svm_j = id_svm_i + 1 : M
        data_train_svm = data_train((data_class == class_type(id_svm_i) | data_class == class_

        switch kernel
        case 'RBF'
        SVM = fitcsvm(data_train_svm(:,1:end-1), data_train_svm(:,end),...
                    'KernelFunction', 'rbf',...
                    'BoxConstraint',C,...
                    'KernelScale',Kernel_Scale,...
                    'ClassNames',[id_svm_j,id_svm_i]);

        case 'Poly'
        SVM = fitcsvm(data_train_svm(:,1:end-1), data_train_svm(:,end),...
                    'KernelFunction', 'polynomial',...
                    'BoxConstraint',1,...
                    'KernelScale', Kernel_Scale, ...
```

```
                    'PolynomialOrder',C, ...
                    'ClassNames',[id_svm_j,id_svm_i]);
        end

        vote_svm = predict(SVM, data_test(:,1:end-1));
        vote = [vote, vote_svm];

    end
end

[predict_label, ~, values_with_freq] = mode(vote, 2);

    % deal with fuzzy cases
    for i = 1:length(values_with_freq)
        if length(cell2mat(values_with_freq(i,1))) ~= 1
            predict_label(i,1) = randi([1,max(class_type)], 1,1);
        end
    end

display('1 vs 1 SVM training and testing completed');

end
```

A.3. Bag of Words vector quantisation pr

```
function [ data_train, data_query] = getData_HQ( MODE, K )
% Generate training and testing data

% Data Options:
%   1. Toy_Gaussian
%   2. Toy_Spiral
%   3. Toy_Circle
%   4. Caltech 101

showImg = 0; % Show training & testing images and their image feature vector (histogram repres

PHOW_Sizes = [4 8 10]; % Multi-resolution, these values determine the scale of each layer.
PHOW_Step = 8; % The lower the denser. Select from {2,4,8,16}

switch MODE
    case 'Toy_Gaussian' % Gaussian distributed 2D points
        %rand('state', 0);
        %randn('state', 0);
        N= 150;
        D= 2;

        cov1 = randi(4);
        cov2 = randi(4);
        cov3 = randi(4);

        X1 = mgd(N, D, [randi(4)-1 randi(4)-1], [cov1 0;0 cov1]);
        X2 = mgd(N, D, [randi(4)-1 randi(4)-1], [cov2 0;0 cov2]);
        X3 = mgd(N, D, [randi(4)-1 randi(4)-1], [cov3 0;0 cov3]);
```

```
    X= real([X1; X2; X3]);
    X= bsxfun(@rdivide, bsxfun(@minus, X, mean(X)), var(X));
    Y= [ones(N, 1); ones(N, 1)*2; ones(N, 1)*3];

    data_train = [X Y];

case 'Toy_Spiral' % Spiral (from Karpathy's matlab toolbox)

    N= 50;
    t = linspace(0.5, 2*pi, N);
    x = t.*cos(t);
    y = t.*sin(t);

    t = linspace(0.5, 2*pi, N);
    x2 = t.*cos(t+2);
    y2 = t.*sin(t+2);

    t = linspace(0.5, 2*pi, N);
    x3 = t.*cos(t+4);
    y3 = t.*sin(t+4);

    X= [[x' y']; [x2' y2']; [x3' y3']];
    X= bsxfun(@rdivide, bsxfun(@minus, X, mean(X)), var(X));
    Y= [ones(N, 1); ones(N, 1)*2; ones(N, 1)*3];

    data_train = [X Y];

case 'Toy_Circle' % Circle

    N= 50;
    t = linspace(0, 2*pi, N);
    r = 0.4
    x = r*cos(t);
    y = r*sin(t);

    r = 0.8
    t = linspace(0, 2*pi, N);
    x2 = r*cos(t);
    y2 = r*sin(t);

    r = 1.2;
    t = linspace(0, 2*pi, N);
    x3 = r*cos(t);
    y3 = r*sin(t);

    X= [[x' y']; [x2' y2']; [x3' y3']];
    Y= [ones(N, 1); ones(N, 1)*2; ones(N, 1)*3];

    data_train = [X Y];

case 'Caltech' % Caltech dataset
    close all;
    imgSel = [15 15]; % randomly select 15 images each class without replacement. (For bot
```

```
folderName = './Caltech_101/101_ObjectCategories';
classList = dir(folderName);
classList = {classList(3:end).name}; % 10 classes

disp('Loading training images...')
% Load Images -> Description (Dense SIFT)
cnt = 1;
if showImg
    figure('Units','normalized','Position',[.05 .1 .4 .9]);
    suptitle('Training image samples');
end
for c = 1:length(classList)
    subFolderName = fullfile(folderName,classList{c});
    imgList = dir(fullfile(subFolderName,'*.jpg'));
    imgIdx{c} = randperm(length(imgList));
    imgIdx_tr = imgIdx{c}(1:imgSel(1));

    for i = 1:length(imgIdx_tr)
        I = imread(fullfile(subFolderName,imgList(imgIdx_tr(i)).name));

        % Visualise
        if i < 6 & showImg
            subaxis(length(classList),5,cnt,'SpacingVert',0,'MR',0);
            imshow(I);
            cnt = cnt+1;
            drawnow;
        end

        if size(I,3) == 3
            I = rgb2gray(I); % PHOW work on gray scale image
        end

        % For details of image description, see http://www.vlfeat.org/matlab/vl_phow.h
        [~, desc_tr{c,i}] = vl_phow(single(I),'Sizes',PHOW_Sizes,'Step',PHOW_Step); %
extracts PHOW features (multi-scaled Dense SIFT)
    end
end

disp('Building visual codebook...')
% Build visual vocabulary (codebook) for 'Bag-of-Words method'
desc_sel = single(vl_colsubset(cat(2,desc_tr{:}), 10e4)); % Randomly select 100k SIFT


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% K-means clustering
% write your own codes here
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cluster_num = K; % K
[C, ~] = vl_kmeans(desc_sel, cluster_num);

%       desc_sel = desc_sel.';
%       [~, C] = kmeans(desc_sel, cluster_num, 'MaxIter', 50, 'Start', 'plus');
```

```
%         [~,C] = kmeans(desc_sel,cluster_num,'MaxIter', 1, 'Start', 'plus');
%         iteration_count = 1;
%
%         [~,C1] = kmeans(desc_sel,cluster_num,'MaxIter',1, 'Start', C);
%         iteration_count = iteration_count + 1;
%
%         while ~isequal(C1,C)
%             C = C1;
%             [idx,C1] = kmeans(desc_sel,cluster_num,'MaxIter',1, 'Start', C);
%             iteration_count = iteration_count + 1;
%         end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        disp('Encoding Images...')
        % Vector Quantisation
        % write your own codes here
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % data_train: [num_data x (dim+1)] Training vectors with class labels
        % data_train(:,end): class labels
        data_train = [];

       for c = 1:length(classList)
           for i = 1:length(imgIdx_tr)
            train_image = desc_tr{c,i};
            % now generate histogram fNN
            [assigned_cluster, ~] = fNN(train_image, C);
            data_tr = zeros(1,cluster_num);

               for j = 1:cluster_num
                   data_tr(1,j) = length(find(assigned_cluster == j));
               end

            data_train = [data_train; data_tr, c];
           end
       end

       display('Training data completed');
        % Clear unused varibles to save memory
        clearvars desc_tr desc_sel
end

switch MODE
    case 'Caltech'
        if showImg
        figure('Units','normalized','Position',[.05 .1 .4 .9]);
        suptitle('Testing image samples');
        end
        disp('Processing testing images...');
        cnt = 1;
        % Load Images -> Description (Dense SIFT)
        for c = 1:length(classList)
```

```
        subFolderName = fullfile(folderName,classList{c});
        imgList = dir(fullfile(subFolderName,'*.jpg'));
        imgIdx_te = imgIdx{c}(imgSel(1)+1:sum(imgSel));

        for i = 1:length(imgIdx_te)
            I = imread(fullfile(subFolderName,imgList(imgIdx_te(i)).name));

            % Visualise
            if i < 6 & showImg
                subaxis(length(classList),5,cnt,'SpacingVert',0,'MR',0);
                imshow(I);
                cnt = cnt+1;
                drawnow;
            end

            if size(I,3) == 3
                I = rgb2gray(I);
            end
            [~, desc_te{c,i}] = vl_phow(single(I),'Sizes',PHOW_Sizes,'Step',PHOW_Step);

        end
    end


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Vector Quantisation
    % write your own codes here
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % data_query: [num_data x (dim+1)] Testing vectors with class labels
    % data_query(:,end): class labels
    data_query = [];
    for c = 1:length(classList)
        for i = 1:length(imgIdx_tr)
            test_image = desc_te{c,i};
            % now generate histogram fNN
            [assigned_cluster, ~] = fNN(test_image, C);
            data_te = zeros(1,cluster_num);

           for j = 1:cluster_num
               data_te(1,j) = length(find(assigned_cluster == j));
           end

        data_query = [data_query; data_te, c];
        end
   end
   display('Testing data completed');

otherwise % Dense point for 2D toy data
    xrange = [-1.5 1.5];
    yrange = [-1.5 1.5];
    inc = 0.02;
```

```
        [x, y] = meshgrid(xrange(1):inc:xrange(2), yrange(1):inc:yrange(2));
        data_query = [x(:) y(:) zeros(length(x)^2,1)];
end
end
```