RANDOMISED DECISION FOREST
Coursework 1 - Selected Topics in Computer Vision

Ahmet Narman
CID: 01578741
MSc in Human & Biological Robotics

Guillaume Dau
CID: 01554132
MSc in Physics with Nanophotonics

1. K-means codebook

For this coursework we were asked to produce an RF classification pipeline for the images of different classes in Caltech 101 dataset using 150 training images and 150 testing images having 10 classes. We start the pipeline by getting the SIFT descriptors from the images using *vl_phow* from the *vlfeat* toolbox. The visual vocabulary was generated by applying MATLAB's k-means clustering algorithm on the SIFT descriptors. For a predetermined dictionary size (k codewords), this algorithm took 100K random descriptors from different images, each having 128 features, and returned the visual dictionary, which are the cluster centers of the k-means algorithm. Initially, the algorithm assigns k cluster centers randomly, and then runs an iterative optimization process to come up with the most optimal cluster centers (codewords) that will have better generalization performance for the given descriptors during the vector quantisation process.

After the dictionary is created, the descriptors of the training and testing images are put through the vector quantisation process resulting in the generation of bag-of-visual-words (BoW). During this process, each descriptor of an image is assigned to its nearest codeword, utimately giving a histogram for each image, whose bins represent the frequency of every codeword in the image. Resulting histograms for the training and testing images are given at Figure 1. Due to visualization purposes, the displayed histogram have a small vocabulary size altering intraclass similarities. Yet, it was seen that images of same classes have relatively similar histograms while images from different classes rather don't. As illustrated in Figure 2, quantisation process' times increases linearly with the codebook size while cobebook generation gets logarithmically more time-consuming with its size. This would imply to choose a rather small vocabulary (200-300 codewords). However, time alone isn't a sufficient selection criteria for choosing the vocabulary size, in fact the global model accuracy strongly dictates this choice. For instance, although replicating the k-means clustering several times with different

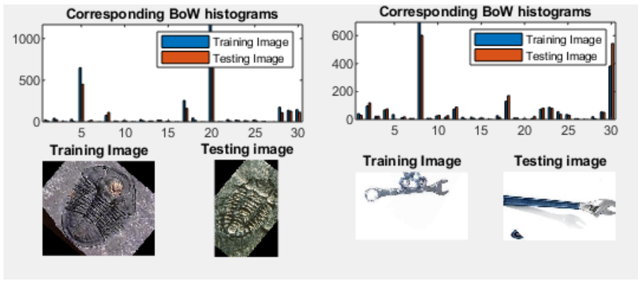

Figure 1: **BoW-representation (histograms) of training and testing images using 30 cluster centers.** **(A)** Trilobite and **(B)** Wrench class samples.

initialization condition (replicates) severely increased the computational time, it did not prove to significantly increase the classifier accuracy. Finally, we selected a codebook of 700 words even though the accuracy gain might not be worth the extra computation time compared to 250 words but the RF classifier optimization demonstrated better results with the bigger codebook.

2. Random Forest classifier

Having obtained a BoW representation for all images, an ensemble of random decision trees is grown using the RF functions given in the assignment (RF_2019/internal). Each tree is trained on a randomly sampled set of BoW-represented training data points having the same size with the original one (Bagging with replacement) and to which class labels are added. (Note that uniqueness of the data points composing each subset is controlled by the bootstrap factor).

As these data sets are pushed through to the trees to be recursively and randomly split into partitioned subsets sharing similar features until they reach leaf nodes, predictions can be made on what class can be found in what leaf of what tree. Thus, when giving unlabeled test data points to the same forest, it acts as a classifier and returns the leaves the data end up in. Using the label probabilities in the leaves,

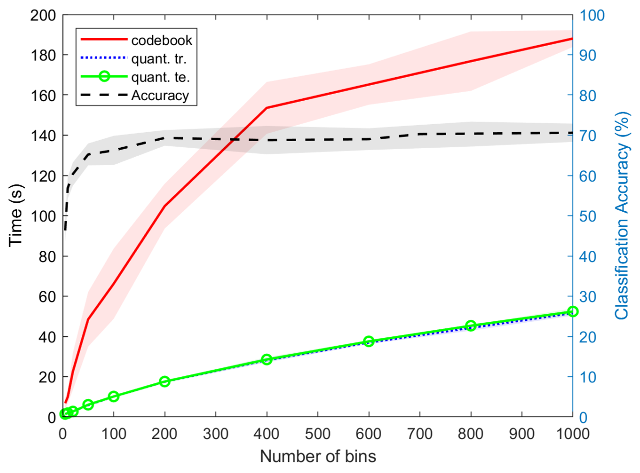

Figure 2: **K-means codebook generation time, Training and testing BoWs times and classification accuracy for different vocabulary sizes** Results averaged on 6 runs. (k_means with 10000 iterations max). RF classification parameters: 300 trees, depth 7, split trials 3, axis aligned weak learner.

class labels are assigned using max. likelihood principle. By comparing predicted classes with the actual classes of the test data, the classification accuracy is obtained. Considered as the overall performance metric of our trained classifier on a particular codebook, it becomes the strongest decision criterion when optimizing the classifier. However, as better recognition performance requires greater computation time and power (higher model complexity), time is also a strong interest depending the real-world application (real-time detection/recognition, identification,..).

The number of trees composing the forest ensures a decorrelation between individual tree predictions. This brings robustness to our classifier by making it less dependent to the data sets of training images that is given to it, and reduces the depth dependent overfitting observed in individual trees. So increasing the number of trees, ensures that individual tree inaccuracies are averaged out to improve performance. Yet, if the amount of trees is too big, the classifier will only demand more computational power and time while the accuracy remaining the same. Thus, for classification, forests of maximum 1000 trees were taken into account during parameter optimization (up to 10000 trees were tested as well but did not show notable improvements). According to experimental runs, the optimal results were obtained for roughly 300 trees. After 300 trees, the classification accuracy did not improve significantly but the training and testing time did.

Tree's depth specifies how much the input data set can be randomly partitioned to exhibit similar features. Shallow depths tend to underfit data because trees don't have enough nodes to recursively split the data into coherently small and class specific pieces. On the contrary, deeper trees - in addition to being more computationally demanding - lead to overfitting because leaf nodes contains fewer data point belonging to specific classes (they belong to specific classes because the objective function splits them this way).Due to the little amount data points available for training and testing, trees don't have to be very deep to be highly discriminative in categorizing classes. A depth of 6-8 suits our classification task while being computationally reasonable.

The number of split trials is the number of attempts we make to randomly split the data arriving at a node to obtain a split maximizing the objective function (information gain). The less trials we make, the data is split less optimally but more randomly. Hence, correlations between trees are suppressed. Therefore, it shouldn't be too high to keep randomness high and for taking advantage of trees variety the most.

Another important factor to consider is the type of weak learner to use. We found a slightly better classification accuracy using axis-aligned weak learner than using 2-pixel weak learner. Although the accuracy change is less than 2%, it is worth noting that the axis aligned test requires less computations as it only compares features of BoW data points to a random threshold on one randomly selected dimension instead of having the difference of two dimensions used in 2-pixel test.

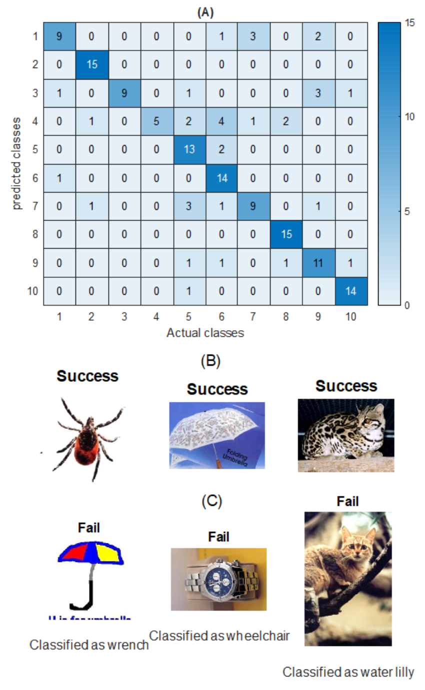

Figure 3: **(A)** Confusion matrix for k-means codebook and RF classifier with optimal parameters. **(B)** Some images that are labeled correctly and **(C)** incorrectly.

The optimization for given parameters contains two steps. The first one consists in finding the number of bins for the K-means clustering that showcase the best classification accuracy on the test data. To do so all classifier parameters was fixed. This process is extremely time consuming - especially for larger cluster numbers, as the whole feature codebook has to be built and the BoW representations have to be generated. After several runs, the best number of bins was chosen to be 700 as illustrated in Figure 2 (This was also chosen because it gave a similarly big dictionary used in RF codebook, which allowed a better discussion).

The second step is a 'grid-search' based process to find the optimum set of parameters for the RF classifier. The achieved top accuracy (averaged on 6 runs) for the RF classifier is of 73.4% with the following hyperparameters: 700 codewords, 300 trees, depth of 6, 8 split trials, Axis aligned weak learner (See Figure 4). The confusion matrix and successful/failed classification examples ar given in Figure 3.

3. Random Forest codebook

Instead of using the unsupervised and generative approach of k-means clustering during the codebook generation, the codebook was generated using the random forest which is a supervised and discriminative way of constructing the dictionary. The other advantage of the RF approach for codebook generation is its speed as random forest requires less calculations and less time during the dictionary generation and BoW generation for each image because of its hierarchical structure and the computational simplicity of its split functions.

During the codebook generation, descriptors of images were fed to the random forest with the class labels of the image they came from. This way, descriptors of certain classes have higher probability of ending up in certain leaves. After the tree training, the images were quantised using the *testTrees_fast* function which gave the corresponding leave indices of descriptors and these were put into the BoW histogram. Generating the BoW for training and testing images, the same RF classification pipeline was used. Yet, as the codebooks were generated in different ways, another parameter optimization was done on RF classifier.

After trying different parameters for the RF codebook in a grid search, the optimum parameters were found to be 6 trees, trees depth of 9 and 3 split trials using the two-pixel test. Also, the two-pixel weak learner performed better on RF classifier, unlike k-means codebook. The corresponding performances of different parameters and two weak learners can be seen in Figure 5.

Compared to the k-means approach, RF codebook approach was much more time-efficient. It was 8 times faster on codebook generation and more than 15 times faster on BoW generation for the optimal parameters. However, the classification accuracy was inferior (almost 10% more error for RF codebook) compared to the k-means codebook model which is a major downside. This accuracy problem may be caused by the descriptors coming from backgrounds instead of objects themselves. These descriptors are not object specific, and they reduce the discriminative performance of the RF codebook. These descriptors (that are general and not unique to a specific class) can be eliminated during the classification process as further improvement.

Finally, we are fully aware that the learning should be validated using a separate validation set, through *n-fold cross validation*. Yet, as the training and testing sets are very small, validation was done on the testing set. It can be argued that Caltech101 dataset has more instances of these images which can be used as real unseen testing data for real-world classification.

(A) K-Means Codebook, RF Classifier (Axis Alligned)				**(B) K-Means Codebook, RF Classifier (2-pixel)**			
Cb Time*:		BoW Time**:		Cb Time:		BoW Time:	
80.12 s		43.53 s		80.12 s		43.53 s	
Tr. Acc.	Tes. Acc.	Tr. Time.	Tes. Time.	Tr. Acc.	Tes. Acc.	Tr. Time.	Tes. Time.
98.6%	**73.4%**	10.4s	**2.6s**	99.3%	72.1%	10.7s	4.1s
(C) RF Codebook (Axis Alligned), RF Classifier (Axis Alligned)				**(D) RF Codebook (2-pixel), RF Classifier (2-pixel)**			
Cb Time:		BoW Time:		Cb Time:		BoW Time:	
10.4 s		**2.67 s**		10.82 s		3.92 s	
Tr. Acc.	Tes. Acc.	Tr. Time.	Tes. Time.	Tr. Acc.	Tes. Acc.	Tr. Time.	Tes. Time.
96.4%	59.2%	10.5s	2.9s	95.7%	64%	10.6	4.1

Figure 4: **Time and accuracy performance of the corresponding classification pipelines.** Results are averaged over 6 runs to eliminate variability. RF classifier is an ensemble of **300 trees** with **depth 6** and **split trials of 8** and a bootstrap factor of **63.2%** for each tree. Vocabulary sizes: 700 codewords for **(A)**,**(B)**, 750 codewords for **(C)**,**(D)** (6 trees, 9 depth, 3 split trials for RF Codebook). * *Codebook generation time (k-means with 1 replicate)* ** *Total vector quantisation time*.

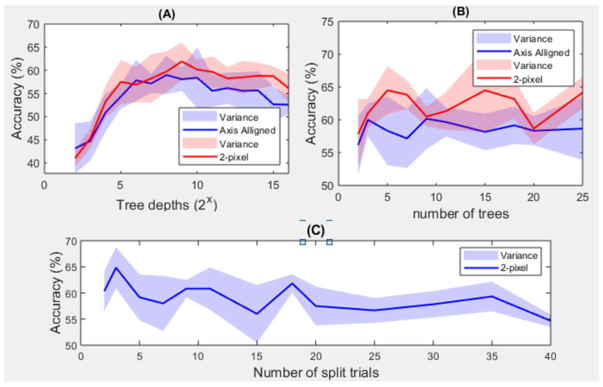

Figure 5: **Influence of different RF codebook parameters on accuracy.** **(A)** trees depth, **(B)** number of trees and **(C)** split trials. Calculated both for axis-aligned and 2-pixel test weak learners (except **(C)**). (RF classifier of 300 trees, 6 depth and 3 splits). Results are averaged over 8 runs.

4. Appendix: Extra Figures and Matlab Codes

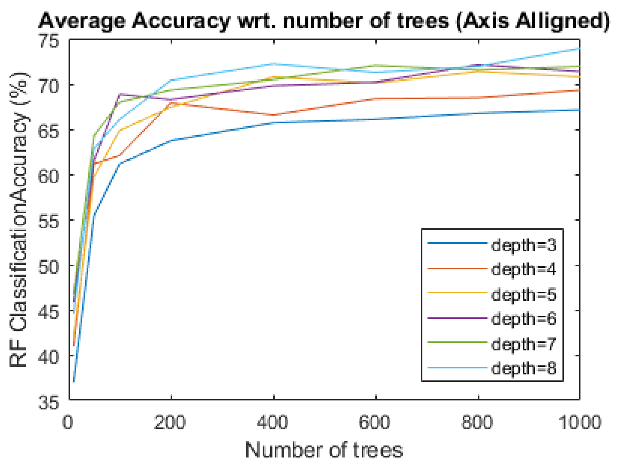

Figure 6: RF classification accuracy with respect to tree numbers and tree depths for K-means codebook. This was put in the appendix because it was a preliminary analysis and does not show the ultimate model. It can be seen that the accuracy increases as depth and tree number increases but after a threshold, the change in accuracy is insignificant.

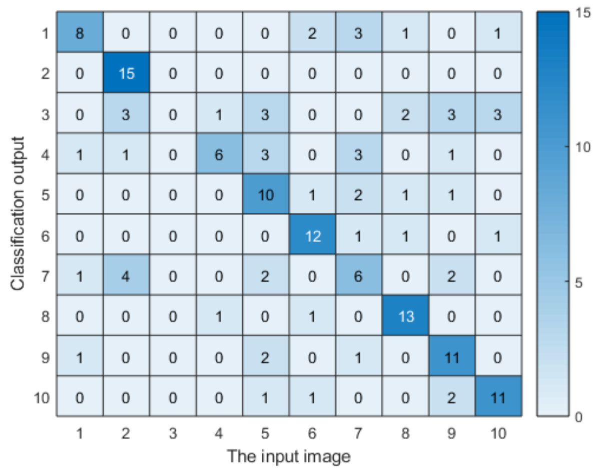

Figure 7: The confusion matrix for RF codebook model. Overall, the performance is worse compared to the K-means model (especially for class 3 for which no correct labeling could be done). However, this is just the confusion matrix of one trained model and different models (even with the same parameters) can give different results.

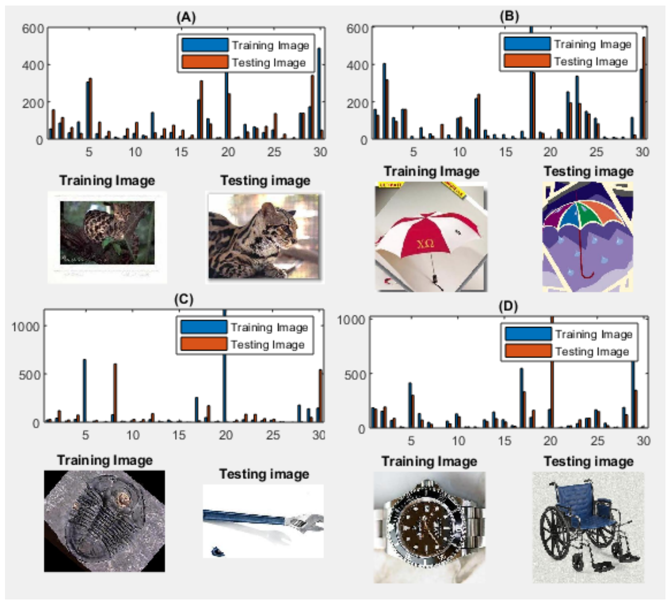

Figure 8: More histogram representation for further discussion. For (A) and (B), the images are from same classes and they have similar histograms. For (C), Images are from different classes and they have different histograms as it should be to allow a robust classification. Yet, checking the worse performing classes on the confusion matrix, it can be seen that on (D), images of different classes can have similar histograms which can result in classification failures. Inherently, images of certain classes have lots of unique descriptors, (having a plain background helps as no descriptors are extracted from background which acts as noise) which helps having better classification accuracy for those classes as it can be seen on the confusion matrix

```
% This code is to be run by section for a fast implementation and
% parameters change of the global image categorisation model.


%% Initialisation

clear all; close all;
init; clc;

%% STEP 1: Getting the data using k-means

numBins =700;

[data_train,data_query]=getData2(numBins);
%%
% If timing the different data processing steps (cobebook generation,
% vector quantisation process) is required.
[ data_train, data_query,t_building_cb,t_quantisation_tr,t_quantisation_te ] =↙
getData2_timing( numBins )

%% STEP 1 prime: Getting the data using random trees

param.num = 30;
param.depth = 10;    % trees depth
param.splitNum = 4; % Number of trials in split function
param.split = 'IG'; % Currently support 'information gain' only


[data_train, data_query] = getData3(param);

%% STEP 2: Reshaping the data for random forest classification

S = size(data_train);
data = [];
for i = 1:S(1)
    % Labels are added to the descriptors
    newData = [reshape(data_train(i,:,:),[S(2),S(3)]) i*ones(S(2),1)];
    % Data matrix is expanded
    data = [data; newData];
end
% training data is ready to be sent to the random forest with labels
data_train = data;

% Repeating the same process for testing data
S = size(data_query);
data = [];
for i = 1:S(1)
    % Labels are added to the descriptors
    newData = [reshape(data_query(i,:,:),[S(2),S(3)]) i*ones(S(2),1)];
    % Data matrix is expanded
    data = [data; newData];
end
% testibg data is ready to be sent to the random forest with labels (labels are↙
discarded when testing)
```

```
data_query = data;

%% STEP 3 Growing classification trees - axis-aligned weak learner

disp('Training RF Classification trees...')

tic

param.num = 300;     % Number of trees
param.depth = 6;     % trees depth
param.splitNum = 3; % Number of trials in split function
param.split = 'IG'; % Currently support 'information gain' only

%Growing the trees and feeding the training data subsets (from bagging step) to them
tree=growTrees(data_train,param);

Training_classifier_time_AA=toc;

%% STEP 3 prime: Growing classification trees - 2-pixel test weak learner

disp('Training RF Classification trees...')

tic

param.num = 10000;     % Number of trees
param.depth = 6;     % trees depth
param.splitNum = 8; % Number of trials in split function
param.split = 'IG'; % Currently support 'information gain' only

%Growing the trees and feeding the training data subsets (from bagging step) to them
tree_2pix=growTrees_2pix(data_train,param); % Only change that is made to growTrees↙
is that
                                            % this function calls splitNode_2pix↙
instead of
                                            % splitNode

Training_classifier_time_2pix=toc;
%% STEP 4 : Classifying through test data - axis-aligned weak learner
tic
disp('Testing RF classification trees...')
% testing the classifier accuracy by classing the test data
labels=testTrees(data_query(:,1:end-1),tree); % Not including labels of testing data
C=[]; % The class labels after classification
for i=1:150
    [~,idx]=max(sum(tree(1).prob(labels(i,:),:)));
    C=[C idx];
end

A= C==data_query(:,end)';
res = 100*sum(A)/150; % Correctness in percentage
% Add a confusion matrix
disp('Classification Accuracy (in percentage): ');
disp(res);
Testing_classifier_time_AA=toc;
```

```
%% STEP 4 : Classifying through test data - 2-pixel test weak learner


disp('Testing RF classification trees...')
tic
% testing the classifier accuracy by classing the test data
labels_2pix=testTrees(data_query(:,1:end-1),tree_2pix); % Not including labels of↙
testing data
C=[]; % The class labels after classification
for i=1:150
    [~,idx]=max(sum(tree(1).prob(labels(i,:),:)));
    C=[C idx];
end

A= C==data_query(:,end)';
res = 100*sum(A)/150; % Correctness in percentage
% Add a confusion matrix
disp('Classification Accuracy (in percentage): ');
disp(res);
Testing_classifier_time_2pix=toc;
```

```
function [ data_train, data_query,t_building_cb,t_quantisation_tr,t_quantisation_te ]↙
= getData2_timing( numBins )
% this function only differs from getData2 by the use of timing output
% arguments: t_building_cb,t_quantisation_tr,t_quantisation_te


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

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}

close all;
imgSel = [15 15]; % randomly select 15 images each class without replacement. (For↙
both training & testing)
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)); % Training image indexes
    imgIdx_te = imgIdx{c}(imgSel(1)+1:sum(imgSel)); % Testing image indexes

    % The loop for extracting descriptors from the training set
    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.↙
```

```
html
        [~, desc_tr{c,i}] = vl_phow(single(I),'Sizes',PHOW_Sizes,'Step',PHOW_Step); %↙
extracts PHOW features (multi-scaled Dense SIFT)
    end

end

%% IMPORTANT PART 1



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

% K-means clustering

% write your own codes here

% Codebook is found using k-means
[codeIndex,codebook] = kmeans(desc_sel', numBins,'maxIter',10000); %if replicates use↙
'replicates',number
t_building_cb=toc;

disp('Encoding Images...')
% Vector Quantisation

% write your own codes here

% Bag of words representation for training data
tic
[R,C]=size(desc_tr);

BOW_tr = zeros(R,C,numBins);

% Assigning each descriptor of each training image to its nearest
% codeword (nearest neighbourg method) to obtain Bag-of-visual-words
% representation of each training image
for i=1:length(classList)
    for j=1:imgSel(1)
        desc_size = size(desc_tr{i,j});
        for k=1:desc_size(2)
            dist = sum((codebook' - double(desc_tr{i,j}(:,k))).^2);
            [~,min_ind] = min(dist);
            BOW_tr(i,j,min_ind) = BOW_tr(i,j,min_ind) + 1;
        end
    end
end

data_train = BOW_tr; % size 10x15xnumBins
t_quantisation_tr=toc;
```

```
%% END OF IMPORTANT PART 1
% Clear unused varibles to save memory
clearvars desc_tr desc_sel



if showImg
figure('Units','normalized','Position',[.05 .1 .4 .9]);
suptitle('Test 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
%suptitle('Testing image samples');
%                   if showImg
%               figure('Units','normalized','Position',[.5 .1 .4 .9]);
%           suptitle('Testing image representations: 256-D histograms');
%           end

% Quantisation

%% IMPORTANT PART 2

tic
BOW_te = zeros(R,C,numBins);

% Assigning each descriptor of each training image to its nearest
% codeword (nearest neighbourg method) to obtain Bag-of-visual-words
% representation of each test image
```

```
for i=1:length(classList)
    for j=1:imgSel(2)
        desc_size = size(desc_te{i,j});
        for k=1:desc_size(2)
            dist = sum((codebook' - double(desc_te{i,j}(:,k))).^2);
            [~,min_ind] = min(dist);
            BOW_te(i,j,min_ind) = BOW_te(i,j,min_ind) + 1;
        end
    end
end

data_query = BOW_te;  % size 10x15xnumBins
t_quantisation_te=toc;


end
```

```
function [node,nodeL,nodeR] = splitNode_2pix(data,node,param)
% Split node code for 2-pixel test weak learners

visualise = 0;

% Initilise child nodes
iter = param.splitNum;
nodeL = struct('idx',[],'t',nan,'dim',[0 0],'prob',[]);
nodeR = struct('idx',[],'t',nan,'dim',[0 0],'prob',[]);

if length(node.idx) <= 5 % make this node a leaf if has less than 5 data points
    node.t = nan;
    node.dim = [0 0];
    return;
end

idx = node.idx;
data = data(idx,:);
[N,D] = size(data);
ig_best = -inf; % Initialise best information gain
idx_best = [];
for n = 1:iter


%%%%%% MODIFIED PART %%%%%%%%


    dim1 = randi(D-1); % Pick first random dimension
    dim2 = randi(D-1); % Pick second random dimension

    while dim1==dim2
        dim2 = randi(D-1); % Pick again second random dimension until the
                           % two randomly picked dimension are different.
    end
    dim=[dim1 dim2];       % Updating the structure of dim compared to the axis-↙
aligned version
    diff_data=data(:,dim1)-data(:,dim2);  %Term by term difference of the data value↙
contained in the 2 dim
    d_min = single(min(diff_data(:,1))) + eps; % Find the data range of this↙
dimension
    d_max = single(max(diff_data(:,1))) - eps;
    t = d_min + rand*((d_max-d_min)); % Pick a random value within the range as↙
threshold
    idx_ = diff_data(:) < t;          % Term by term comparison of the difference↙
vector to the threshold


%%%%%% END OF MODIFIED PART %%%%%%%%%%%%%%


    ig = getIG(data,idx_); % Calculate information gain


    if (sum(idx_) > 0 & sum(~idx_) > 0) % We check that children node are not empty
```

```
        [node, ig_best, idx_best] = updateIG(node,ig_best,ig,t,idx_,dim,idx_best);
    end

end

nodeL.idx = idx(idx_best);
nodeR.idx = idx(~idx_best);

if visualise
    visualise_splitfunc(idx_best,data,dim,t,ig_best,0)
    fprintf('Information gain = %f. \n',ig_best);
    pause();
end

end

function ig = getIG(data,idx) % Information Gain - the 'purity' of data labels in↙
both child nodes after split. The higher the purer.
L = data(idx);
R = data(~idx);
H = getE(data);
HL = getE(L);
HR = getE(R);
ig = H - sum(idx)/length(idx)*HL - sum(~idx)/length(idx)*HR;
end

function H = getE(X) % Entropy
cdist= histc(X(:,1:end), unique(X(:,end))) + 1;
cdist= cdist/sum(cdist);
cdist= cdist .* log(cdist);
H = -sum(cdist);
end

function [node, ig_best, idx_best] = updateIG(node,ig_best,ig,t,idx,dim,idx_best) %↙
Update information gain
if ig > ig_best
    ig_best = ig;
    node.t = t;
    node.dim = dim;
    idx_best = idx;
else
    idx_best = idx_best;
end
end
```

```
function label = testTrees_2pix(data,tree)
% 2-PIXEL TEST VERSION OF testTrees
% Slow version - pass data point one-by-one


cc = [];
for T = 1:length(tree)
    for m = 1:size(data,1);
        idx = 1;

        while tree(T).node(idx).dim
            t = tree(T).node(idx).t;
            dim = tree(T).node(idx).dim;
            % Decision

%%%%%% MODIFIED PART %%%%%%%%%%
            if data(m,dim(1))-data(m,dim(2)) < t % Pass data to left node
%%%%%% END OF MODIFED PART %%%%%%%%%%%
                idx = idx*2;
            else
                idx = idx*2+1; % and to right
            end

        end
        leaf_idx = tree(T).node(idx).leaf_idx;

        if ~isempty(tree(T).leaf(leaf_idx))
            p(m,:,T) = tree(T).leaf(leaf_idx).prob;
            label(m,T) = tree(T).leaf(leaf_idx).label;

%              if isfield(tree(T).leaf(leaf_idx),'cc') % for clustering forest
%                  cc(m,:,T) = tree(T).leaf(leaf_idx).cc;
%              end
        end
    end
end

end
```