16720J: Homework 2 - Bag-of-Words for Scene Classification

Name - WENBO ZHAO (WZHAO1)

October 8, 2015

3 Warming up with some theory (10pts)

Question 3.1 (3pts, 2-3 lines)

Given an $N \times N$ image, and an $h \times h$ Gaussian filter, we can convolve the image and the filter in $O(h)$ time by first convolving the image horizontally with an $h \times 1$ filter and then convolving vertically with a $1 \times h$ filter.

Question 3.2 (2pts, 2-3 lines)

No. Because *bag-of-words* takes the features of images (by filtering or feature points extraction) and clusters them into a visual words dictionary. The representation of images is converted to mapping each pixel to the dictionary. After histing the frequency of each visual word in the mapped images, we get totally statistic features, neglecting their spatial distribution information.

Question 3.3 (5 pts, 2-3 lines)

Because we want to use a bank of different filters to extract the local texture information of an image in a sampling-reasonable way, but at the same time depressing the noise as much as possible.

4 Representing the World with Visual Words (40pts)

Question 4.1 (5 pts, 3-4 lines) Theory:

There are 3 groups of filters, 11 in each group. For one group:

The first 3 filters are "*Gaussian*" filters, they smooth the noise in the image.

The second 4 filters are "*LoG*" filters, they smooth the noise and detect the edges in the image.

The third 4 filters are "*Derivative of Gaussians*" filters, they pick up the "*gradient change in a direction*" in the image.

Question 4.2 (15 pts)

Coding question, put your implementation in **baseline/getFilterBankAndDictionary.m**

```
function [filterBank,dictionary] = getFilterBankAndDictionary(trainFiles)
% This function generates a dictionary given a list of images
% - INPUT: * trainFiles: a cell array of strings containing the full path to all images
% - OUTPUTS: * filterBank: a cell array of filters from FUNC::createFilterBank
%            * dictionary: a visual words dictionary from FUNC::kmeans
%
% Author: WENBO ZHAO (wzhao1@andrew.cmu.edu)
% Date: Oct 3, 2015
% Log: (v0.1)-(first draft, written all the functions)-(Oct 3, 2015)
%      (v0.2)-(modified: fixed bug: improved: )
%
%%
% debug = 'finished filter response, start k-means'
% debug = 'start filter response';
%% Set directories
% imageDir = '../images/'
% switch debug
%     case 'start filter response'
%% generate filter bank
fprintf('Getting filter bank ... \n');
filterBank = createFilterBank();
fprintf('Done.\n');
%% Generate filter responses
fprintf('Generating filter responses ... \n');
alpha = 100; % [50,150]
for i = 1:length(trainFiles)
    I = imread(trainFiles{i});
    filterResp = extractFilterResponses(I, filterBank);
    randPixels = randperm(alpha);
    filterResp = filterResp(randPixels, :);
    filterResponses(i,:) = filterResp(:);
end
filterResponses = reshape(filterResponses, [length(trainFiles)*alpha, 3*size(filterBank,
fprintf('saving filter responses ... \n');
save('filterResponses', 'filterResponses');
fprintf('Done.\n');

%%     case 'finished filter response, start k-means'
% load filterBank.mat
% load filterResponses.mat
%% Cluster filter response
fprintf('Getting dictionary ... \n');
K = 200; % [100, 300]
[~,dictionary] = kmeans(filterResponses, K, 'EmptyAction', 'drop');
fprintf('Saving dictionary ... \n');
save('dictionary', 'dictionary');
fprintf('Done.\n');
% end

```

```
end
```

Question 4.3 (5 pts, 3-4 lines) Theory

If the visual words dictionary is too small, it cannot fully represent the filtered responses of images in the training set, thus cannot distinguish the filtered responses well. Contrarily, if the dictionary goes too large, similar responses can be represented by different words, it lacks generalization and is sensitive to noise. It also yields high mapping dimension and exhaustive computation.

Question 4.4 (15 pts)

Coding question, put your implementation in `baseline/getVisualWords.m`

```
function [wordMap]=getVisualWords(I,filterBank,dictionary)
  % This function map each pixel in the image to its closest word in the
  % dictionary.
  % - INPUTS: * I: image
  %           * filterBank: a bank of filters from FUNC::createFilterBank
  %           * dictionary: a visual words dictionary from FUNC::kmeans
  % - OUTPUT: * wordMat: an index map of the size(I) that maps each pixel
  %             response to its closest in the dictionary
  %
  % Author: WENBO ZHAO (wzhao1@andrew.cmu.edu)
  % Date: Oct 3, 2015
  % Log: (v0.1)-(first draft, written all the functions)-(Oct 3, 2015)
  %      (v0.2)-(modified: fixed bug: improved: )
  %
fprintf('\nMapping the filtered response of each pixel in image to its closest index in

% the filtered response for Image
% alpha = 100;
I_response = extractFilterResponses(I, filterBank); % response of (M pixels * N filters)
% compute the distance(ImageResponse, dictionary)
wordMap = zeros(size(I_response, 1),1);
dist = pdist2(I_response, dictionary);
[~, wordMap] = min(dist,[],2);
% for i = 1:size(I_response, 1)
%     dist = pdist2(I_response(i,:), dictionary);
%     [~, wordMap(i)] = min(dist); % find the closest and assign the index
% end
wordMap = reshape(wordMap, [size(I,1), size(I,2)]);

end
```

5 Building a Recognition System (95pts)

Question 5.1 (10 pts)

Coding question, put your implementation in `baseline/getImageFeatures.m`

```
function [h] = getImageFeatures(wordMap,dictionarySize)
% This function returns the histogram of visual words (bag of words) within
% the given image.
% - INPUTS: * wordMap: an index map of the size(Image) that maps each pixel
%             response to its closest in the dictionary
%           * dictionarySize: # of visual words in the dictionary
% - OUTPUT: * h: a (dictionarySize*1) histogram
%
% Author: WENBO ZHAO (wzhao1@andrew.cmu.edu)
% Date: Oct 4, 2015
% Log: (v0.1)-(first draft, written all the functions)-(Oct 4, 2015)
%      (v0.2)-(modified: fixed bug: improved: )
%

h = hist(wordMap(:), dictionarySize);
h = (h./sum(h)).'; % normalize, sum(h)=1

end
```

Question 5.2 Extra credit (5 pts 2-3 lines) Theory:

Because by normalizing all histograms by the total number of features in the image, we first make the histograms from different sublayers and subcells comparable, and we also reduce computational cost.

Question 5.3 Extra credit (5 pts 2-3 lines) Theory

Because by weighting different layers inversely proportional to the cell size at the layer, we penalize matches found in larger cells that might induce increasingly dissimilar features.

Question 5.4 (20 pts)

Coding question, put your implementation in `baseline/getImageFeaturesSPM.m`

```
function [h] = getImageFeaturesSPM(layerNum, wordMap, dictionarySize)
% This function forms a multi-resolution representation of the given image
% - INPUTS: * layreNum: # of layers, L+1, [0 1 2 ... L]
%           * wordMap: an index map of the size(Image) that maps each pixel
%             response to its closest in the dictionary
%           * dictionarySize: # of visual words in the dictionary, K
%             clusters
% - OUTPUT: * h: a ( (K(4^(L+1)-1)/3 * 1) histogram
```

```
%
% Author: WENBO ZHAO (wzhao1@andrew.cmu.edu)
% Date: Oct 4, 2015
% Log: (v0.1)-(first draft, written all the functions)-(Oct 4, 2015)
%      (v0.2)-(modified: fixed bug: improved: )
%

wordMap = wordMap(:);
%% number of cells for each layer
numCell = ones(layerNum,1);
for i = 1:layerNum
    numCell(i) = (2^(i-1))^2;
end
%% index into each cell of the wordMap in each layer
index = [];
% for i = 1:layerNum
%     index(:,i)= [0:length(wordMap)./numCell(i):length(wordMap)]; % e.g.
%     % [0 19200 38400 57600 76800] for length(wordMap)=76800 and numCell=4
% end
index = [0:length(wordMap)./numCell(layerNum):length(wordMap)];
%% start from histing the finest layer (L+1)
h_2 = [];
for j = 1:numCell(layerNum)
    ind = [floor(index(j)+1) : 1 : floor(index(j+1))]; % index for the jth cell
%     h_2 = [h_2; hist(wordMap(ind), dictionarySize).']; % h of size(numCell(layerNum) *
    h_2 = [h_2; (getImageFeatures(wordMap(ind), dictionarySize)).'];
end
%% A temporal solution for coarser layers
h_2 = reshape(h_2, [numCell(layerNum), dictionarySize]);
h_1 = [sum(h_2(1:4,:), 1); sum(h_2(5:8,:), 1); sum(h_2(9:12,:), 1); sum(h_2(13:16,:), 1)
h_1 = bsxfun(@rdivide, h_1, sum(h_1,2)); %
h_0 = [sum(h_2(:,:), 1)];
h_0 = h_0./sum(h_0); %
%%
%% weight
weight = 1/2.*ones(layerNum,1);
weight(1) = 2^(-(layerNum-1));
weight(2) = 2^(-(layerNum-1));
for i = 3:layerNum
    weight(i) = 2^((i-1)-(layerNum-1)-1);
end
%% sum the weighted hist and normalize
h = [weight(1).*h_0; weight(2).*h_1; weight(3).*h_2];
h = h(:);
h = sqrt(h); %
h = h./norm(h,1);

end
```

Question 5.5 (10 pts)

Coding question, put your implementation in `baseline/distanceToSet.m`

```
function [histInter] = distanceToSet(wordHist, histograms)
% This function returns the histogram intersection similarity between
% wordHist and histograms.
% - INPUTS: * wordHist: a (K(4^(L+1)-1)/3 * 1) histogram
%           * histogram: a (K(4^(L+1)-1)/3 * T) histogram from T training
%             samples
% - OUTPUT: histInter: a (1*T) vector of similarity
%
% Author: WENBO ZHAO (wzhao1@andrew.cmu.edu)
% Date: Oct 4, 2015
% Log: (v0.1)-(first draft, written all the functions)-(Oct 4, 2015)
%      (v0.2)-(modified: fixed bug: improved: )
%
histInter = sum( bsxfun(@min, wordHist, histograms) ); % check dimensions
% must be (K(4^(L+1)-1)/3 * 1) and (K(4^(L+1)-1)/3 * T), histInter is (1*T)

end
```

Question 5.6 (5 pts)

Coding question, put your implementation in **baseline/createHistograms.m**

```
function outputHistograms = createHistograms(dictionarySize,imagePaths,wordMapDir)
%code to compute histograms of all images from the visual words
%imagePaths: a cell array containing paths of the images
%wordMapDir: directory name which contains all the wordmaps

outputHistograms = []; %create a NumImage x histogram matrix of histograms.
                       %this variable will store all the histograms of training images


%
layerNum = 3;
for i = 1:length(imagePaths)
     fprintf('Outputing histogram %d: %s\n', i, imagePaths{i});
     wordMap = load(fullfile(wordMapDir, [strrep(imagePaths{i},'.jpg','.mat')]));
     wordMap = wordMap.wordMap;
     outputHistograms = [outputHistograms, getImageFeaturesSPM(layerNum, wordMap, dictio
end

end
```

Question 5.7 (5 pts)

Coding question, provide any helper code you wrote for this section and list the files here. Also make sure your `.mat` is included in your submission.

```
% Ishan Misra
% CV Fall 2014 - Provided Code
```

```
%main script to train your system.
%for debugging, comment out function calls that you have run already

load('traintest.mat');
imageDir = '../images'; %where all images are located
targetDir = '../wordmap';%where we will store visual word outputs



%%compute filter responses and make dictionary
fprintf('Computing dictionary ... \n');
computeDictionary(trainImagePaths,imageDir);
load('dictionary.mat','dictionary','filterBank');
fprintf('done\n');

%%now compute visual words for each image
numCores = 2; %number of parallel jobs to run. Laptops may not support higher numbers
fprintf('Computing visual words ... ');
batchToVisualWords(trainImagePaths,classnames,filterBank,dictionary,imageDir,targetDir,
fprintf('done\n');

%%now compute histograms for all training images using visual word files
tic
trainingHistogramsFile = fullfile(targetDir,'trainingHistograms.mat');
dictionarySize = size(dictionary,1);
fprintf('Computing histograms ... \n');
trainHistograms = createHistograms(dictionarySize,trainImagePaths,targetDir);
fprintf('done\n');
toc
save(trainingHistogramsFile,'trainHistograms');
load(trainingHistogramsFile,'trainHistograms');


%%the test code just needs to load trainOutput.mat, so lets store it
save('trainOutput.mat','dictionary','filterBank','trainHistograms','trainImageLabels');
```

Question 5.8 (10 pts)

Coding question, put your implementation in `baseline/knnClassify.m`

```
function [predictedLabel] = knnClassify(wordHist,trainHistograms,trainingLabels,K, choic
%This function
% - INPUTS: * wordHist: a (K(4^(L+1)-1)/3 * 1) histogram
%           * trainHistograms: a (K(4^(L+1)-1)/3 * T) histogram from T
%           training samples
%           * trainingLabels: a T*1 labels for training set
%           * K: (K) nearest neighbors
% - OUTPUT: * predictedLabel: predicted label for new wordHist
%
% Author: WENBO ZHAO (wzhao1@andrew.cmu.edu)
% Date: Oct 5, 2015
% Log: (v0.1)-(first draft, written all the functions)-(Oct 5, 2015)
```

```
%       (v0.2)-(modified: fixed bug: improved: )
%
%% find the K nearest neighbors of wordHist in trainHistograms
% if ~exist(choiceDistance)
%      choiceDistance = 'similarity';
% end
disp(choiceDistance);
% choiceDistance = 'similarity' % might want use PARSE??
[~, neighbor] = topKNeighbor(wordHist,trainHistograms, trainingLabels, K, choiceDistance
%% predict the label for wordHist
predictedLabel = mode(neighbor);
% predictedLabel = predictLabel(neighbor, max(trainingLabels));

end

function [distance, neighbor] = topKNeighbor(testSet, trainSet, trainLabel, K, choiceDis
% This function returns the top K distance of testSet to trainSet and their
% labels.
%
%% choice of distance
% default choice
% if ~exist(choiceDistance)
%      choiceDistance = 'similarity';
% end

switch lower(choiceDistance)
    case 'similarity'
        % - similarity
        dist = distanceToSet(testSet, trainSet); % similarity in this case
        [dist_sort, pos]= sort(dist, 'descend'); % sort the distance in descending order
        distance = dist_sort(1:K); % find the top K distances...
        neighbor = trainLabel(pos(1:K)); % and their corresponding labels in trainSet
    case 'euclid'
        % - Euclid distance
        dist = pdist2(testSet.', trainSet.', 'euclid');
        [dist_sort, pos]= sort(dist, 'ascend');
        distance = dist_sort(1:K);
        neighbor = trainLabel(pos(1:K));
    case 'cosine'
        % - Cosine distance
        dist = pdist2(testSet.', trainSet.', 'cosine');
        [dist_sort, pos]= sort(dist, 'ascend');
        distance = dist_sort(1:K);
        neighbor = trainLabel(pos(1:K));
    case 'kldivergence'
        % - K-L divergence
        KL = @(X, Y)(sum(bsxfun(@times, X, bsxfun(@minus, log2(X),log2(Y)))));
        dist = pdist2(testSet,trainSet, @(testSet,trainSet) KL(testSet,trainSet));
        [dist_sort, pos]= sort(dist, 'ascend');
        distance = dist_sort(1:K);
        neighbor = trainLabel(pos(1:K));
    case 'builtinknn'
        [idx, dist] = knnsearch(trainSet.',testSet.', 'dist','euclid','k',K);
        distance = dist;
```

```
        neighbor = trainLabel(idx);
    otherwise
        disp('Undefined distance!\n');
end
end
```

Question 5.9 (10 pts)

Coding question, put your implementation in `evaluateRecognitionSystem.m`

```
%Loading the dictionary, filters and training data
numCores=2;
imageDir = '../images'; %where all images are located
targetDir = '../wordmap';%where we will store visual word outputs
load('traintest.mat');
load('trainOutput.mat');

% Close the pools, if any
% try
%     fprintf('Closing any pools...\n');
%     matlabpool close;
% catch ME
%     disp(ME.message);
% end
% fprintf('Will process %d files in parallel to compute visual words ...\n',length(trai
% fprintf('Starting a pool of workers with %d cores\n', numCores);
% matlabpool('local',numCores);

% predict
k = 5; % odd number to avoid tie
layerNum = 3;
distance = 'similarity'
% distance = 'euclid'
% distance = 'cosine'
% distance = 'kldivergence'
% distance = 'builtinKnn'
C = zeros(length(unique(trainImageLabels)));
tic
for i = 1:length(testImagePaths)
    image = imread(fullfile(imageDir, testImagePaths{i}));
    wordMap = getVisualWords(image, filterBank, dictionary);
    h = getImageFeaturesSPM( layerNum, wordMap, size(dictionary,1));
    predictedLabel = knnClassify(h, trainHistograms, trainImageLabels, k, distance);
    fprintf('The %d th image, Label: true: %d, predict: %d \n', i, testImageLabels(i), p
    C(predictedLabel,testImageLabels(i)) = C(predictedLabel,testImageLabels(i)) + 1;
end
toc
% accuracy
fprintf('Confusion matrix:');
C
accuracy = trace(C)/sum(C(:))
```

Verbatim copy of your confusionMatrix:

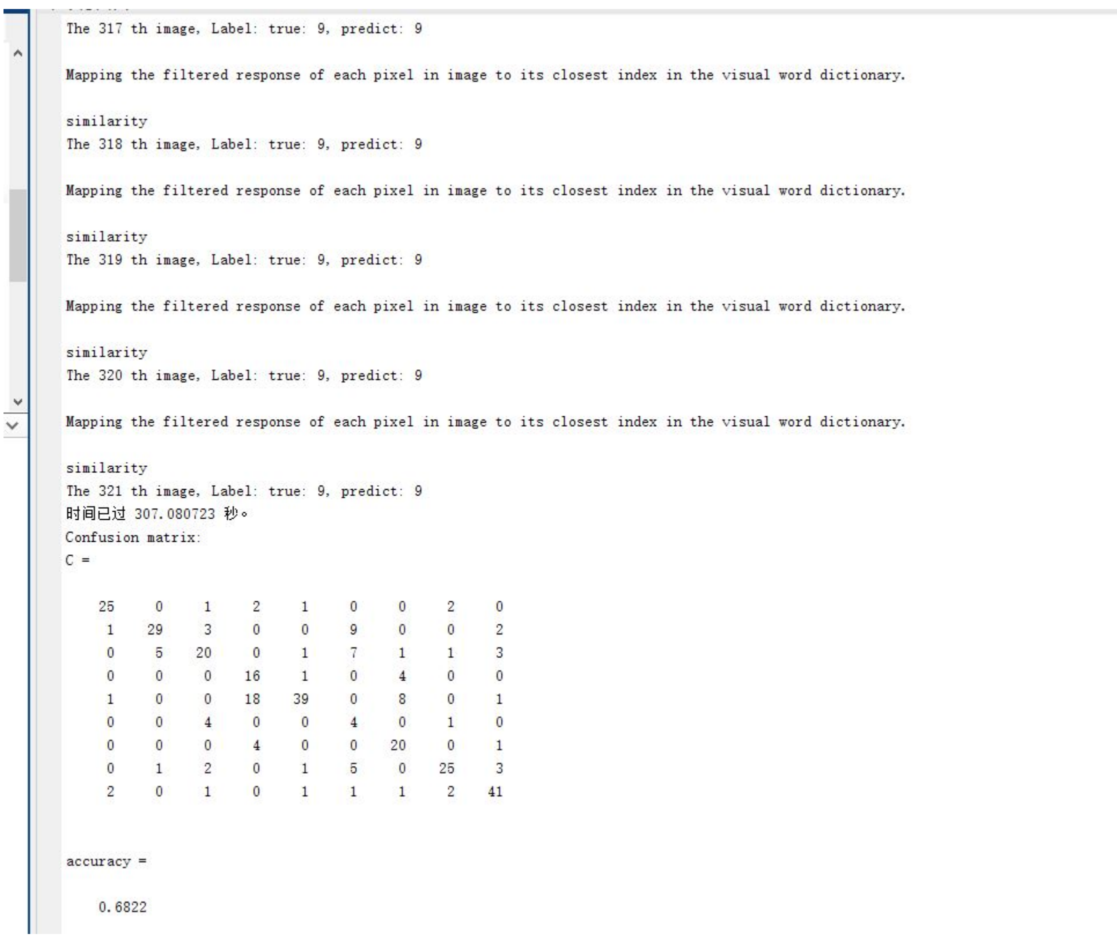
```
K = 1, similarity
    24     0     0     0     0     0     1     1     0
     0    19    12     2     1     7     1     1     3
     0     9    11     1     1     4     0     3     5
     1     4     4    28     9     1     7     1     1
     2     0     0     5    30     0     5     3    10
     1     3     2     2     0     8     0     7     2
     0     0     0     2     2     1    19     0     0
     0     0     2     0     1     4     0    13     3
     1     0     0     0     0     1     1     2    27

K = 5, similarity
    25     0     0     2     1     0     0     1     0
     0    24    16     2     0    11     1     3     4
     0     7     9     1     0     3     0     3     4
     1     1     2    26     6     0    10     2     0
     2     0     0     6    36     0     3     2     6
     0     3     2     0     0     6     0     7     2
     1     0     0     3     1     1    19     0     0
     0     0     1     0     0     4     0    12     3
     0     0     1     0     0     1     1     1    32

K = 11, similarity
    27     0     1     0     0     0     0     1     0
     0    26    16     1     0    12     1     3     5
     0     5    10     1     0     7     0     2     4
     0     1     2    29     4     0    12     2     0
     2     0     0     7    40     0     4     2     5
     0     3     0     0     0     5     0     7     1
     0     0     0     2     0     1    16     0     1
     0     0     1     0     0     1     0    13     2
     0     0     1     0     0     0     1     1    33

K = 15, similarity
    27     0     0     0     0     0     0     0     0
     0    25    13     1     0    13     2     1     4
     0     6    15     0     0     8     0     6     4
     0     2     1    29     4     0    10     3     0
     2     0     0     8    40     0     6     4     3
     0     2     1     0     0     2     0     5     1
     0     0     0     2     0     1    16     0     1
     0     0     0     0     0     2     0    11     2
     0     0     1     0     0     0     0     1    36
```

```
K = 21, similarity
    24     0     0     0     0     0     0     0     0
     0    23    12     2     0    10     2     3     5
     0     5    13     0     1     5     0     4     3
     0     1     4    28     2     0     9     3     0
     5     0     0    10    41     1     5     3     3
     0     6     0     0     0     8     0     5     1
     0     0     0     0     0     1    17     0     1
     0     0     1     0     0     0     0    11     2
     0     0     1     0     0     1     1     2    36
```

Question 5.10 (10 pts)

LAST MINUTE UPDATE: ACCURACY TO 0.6822

Figure 1: **LAST MINUTE UPDATE**



```
The 317 th image, Label: true: 9, predict: 9

Mapping the filtered response of each pixel in image to its closest index in the visual word dictionary.

similarity
The 318 th image, Label: true: 9, predict: 9

Mapping the filtered response of each pixel in image to its closest index in the visual word dictionary.

similarity
The 319 th image, Label: true: 9, predict: 9

Mapping the filtered response of each pixel in image to its closest index in the visual word dictionary.

similarity
The 320 th image, Label: true: 9, predict: 9

Mapping the filtered response of each pixel in image to its closest index in the visual word dictionary.

similarity
The 321 th image, Label: true: 9, predict: 9
时间已过 307.080723 秒。
Confusion matrix:
C =

    25     0     1     2     1     0     0     2     0
     1    29     3     0     0     9     0     0     2
     0     5    20     0     1     7     1     1     3
     0     0     0    16     1     0     4     0     0
     1     0     0    18    39     0     8     0     1
     0     0     4     0     0     4     0     1     0
     0     0     0     4     0     0    20     0     1
     0     1     2     0     1     5     0    25     3
     2     0     1     0     1     1     1     2    41


accuracy =

    0.6822
```

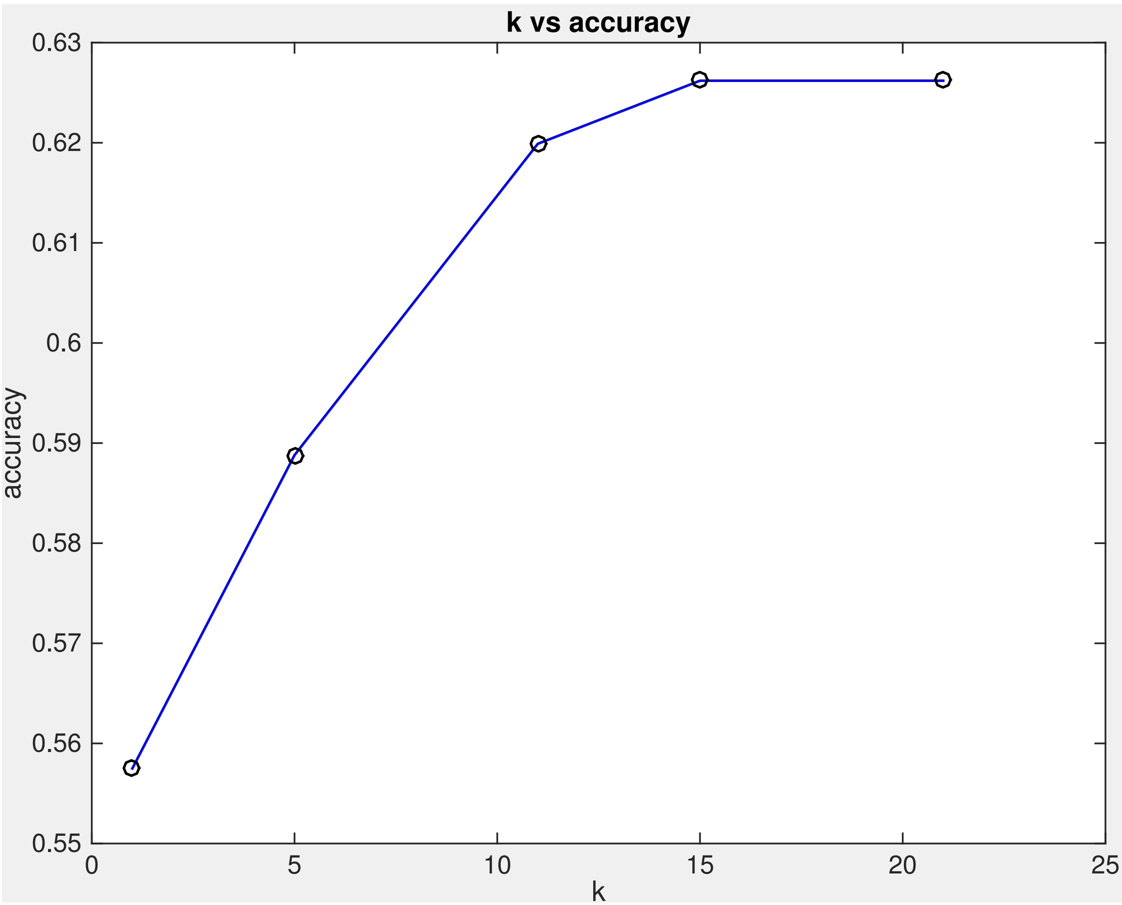
We found when k increases, the accuracy also increases. But when k gets large, the accuracy no longer changes.

A plot of k *v.s.* the classification accuracy (see Fig. 2).

K	1	5	11	15	21	25
Accuracy	0.5575	0.5888	0.6199	0.6262	0.6262	**0.6822**

Table 1: **Accuracy versus k values.** Add some nice observations here!

Figure 2: k *v.s.* the classification accuracy



Results of classification (see Fig. 3).

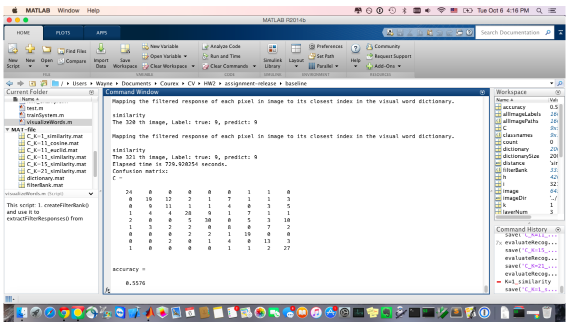
Question 5.11 (5 pts, 2-3 lines): Theory

Nearest neighbor method needs to compute the similarity of data vectors in the dataset. For a large dataset, we need to initialize many centers, and if the vector dimension is high, the computational cost increases. Besides, it does not uniquely weigh different distances.

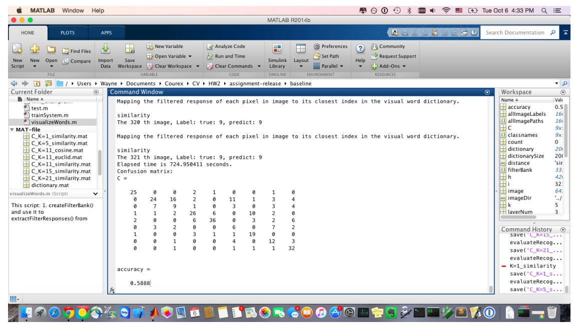
6 Final thoughts on visual words (40 pts)

Question 6.1 (20 pts)

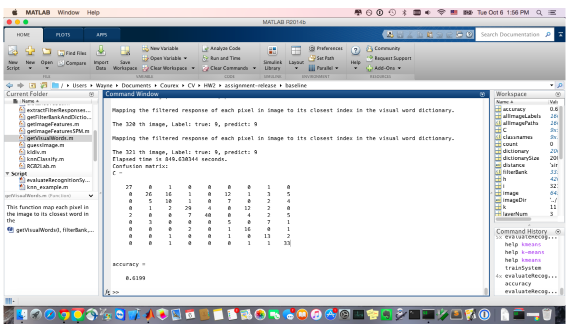
Coding question, put your implementation in `visualizeWords.m`



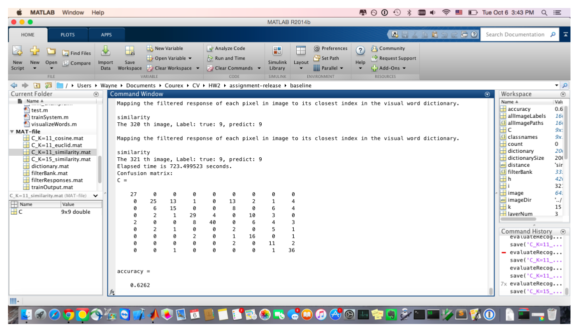
k=1



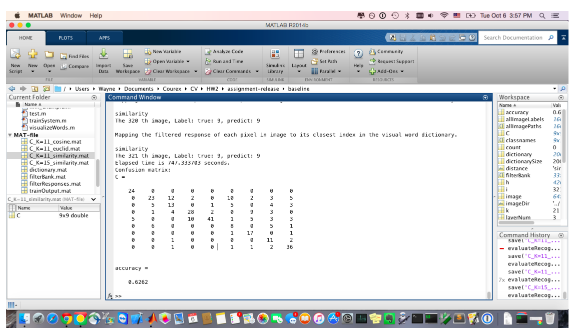
k=5



k=11



k=15



k=21

Figure 3: Results of classification.

```
% This script: finds the 9*9 pixel patch in which the center pixel
% corresponds to the index mapping to the visual word in dictionary.
%
% Author: WENBO ZHAO (wzhao1@andrew.cmu.edu)
% Date: Oct 6, 2015
% Log: (v0.1)-(first draft, written all the functions)-(Oct 6, 2015)
%      (v0.2)-(modified: fixed bug: improved: )
%
% clear all, close all, clc
%
% set directories
addpath ./export_fig
imageDir = '../images';
wordMapDir = '../wordmap';
% set useful variables
load traintest.mat
load('dictionary.mat','dictionary');
numWords = size(dictionary,1); % number of visual words
numPixelInPatch = 9; % numPixelInPatch^2 pixels in a patch
% find the pixel patch of visual words in the training image set
for i = 1:numWords
    pixelPatch{i} = zeros(numPixelInPatch, numPixelInPatch, 3);
end
count = zeros(1,numWords);
fprintf('Finding pixel patches for each visual word...\n');

for i = 1:length(trainImagePaths)
    image = imread(fullfile(imageDir,trainImagePaths{i}));
    word = load(fullfile(wordMapDir, [strrep(trainImagePaths{i}, '.jpg','.mat')]));
    wordMap = word.wordMap;

    for i=(4+1):(size(wordMap,1)-4)
        for j=(4+1):(size(wordMap,2)-4)
            patch = image( (i-4):(i+4), (j-4):(j+4), : ); % read the image patch
            dictInd = wordMap(i,j); % corresponding visual word index for the current pi
            pixelPatch{dictInd} = pixelPatch{dictInd} + double(patch);
            count(dictInd) = count(dictInd) + 1;
        end
    end

end

% average
for i = 1:numWords
    if pixelPatch(i) > 0
        avePixelPatch{i} = uint8(pixelPatch{i} ./ count(i));
    end
end

fprintf('Saving pixel patches...\n');
save('pixelPatch', 'pixelPatch');
fprintf('Done.\n');
```

```
% visualize
imdisp(avePixelPatch);
```

Add your figure here. See Fig. 4.

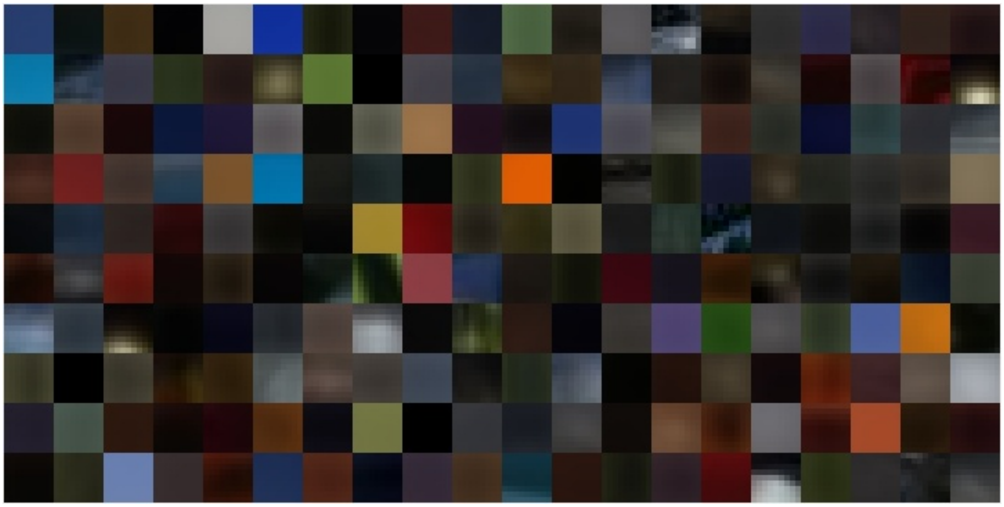


Figure 4: **Visualization of words.** The visualization of one word is essentially searching all the images for pixel patches that have their center indexes mapped to this word. The basis idea is that for a pixel in the image that is categorized to a word in the dictionary, its surrounding pixels should at least close to this pixel. So summing up and average all the patches will give us a visual sense of what the word will look like.

Question 6.2 (10 pts)

Add your explanation here with some nice pictures.
Explanation: For the demonstration purpose, I choose three classes from the pool to visualize: bamboo, basilica and railroad. The classification accuracy (as shown in **Question 5.9**) for class *bamboo* and *basilica* is high, while for *railroad* is low, and *railroads* are much wrongly classified as *basilica*. If we want high classification accuracy, we expect the words within class are similar while across classes are distinctive. We can see from Fig. 5, that for *bamboo*, its top 10 words looks similar. But for the latter two their top 10 words looks like they have somewhat unique patterns, especially for *railroad*. Looking in the detail, we found for *railroads* they have more word representations than others. This is due to the image set for *railroad* has many distinctive changes (light, surroundings, etc). These words are of less representation to this class, but might more to classes that share similar words, say, *basilica*. To overcome this problem, we can on the one hand enlarge the dataset or capture new features for this class to extract more generalized patterns, on the other hand reduce the cluster number of this class to avoid sensitiveness to noise.

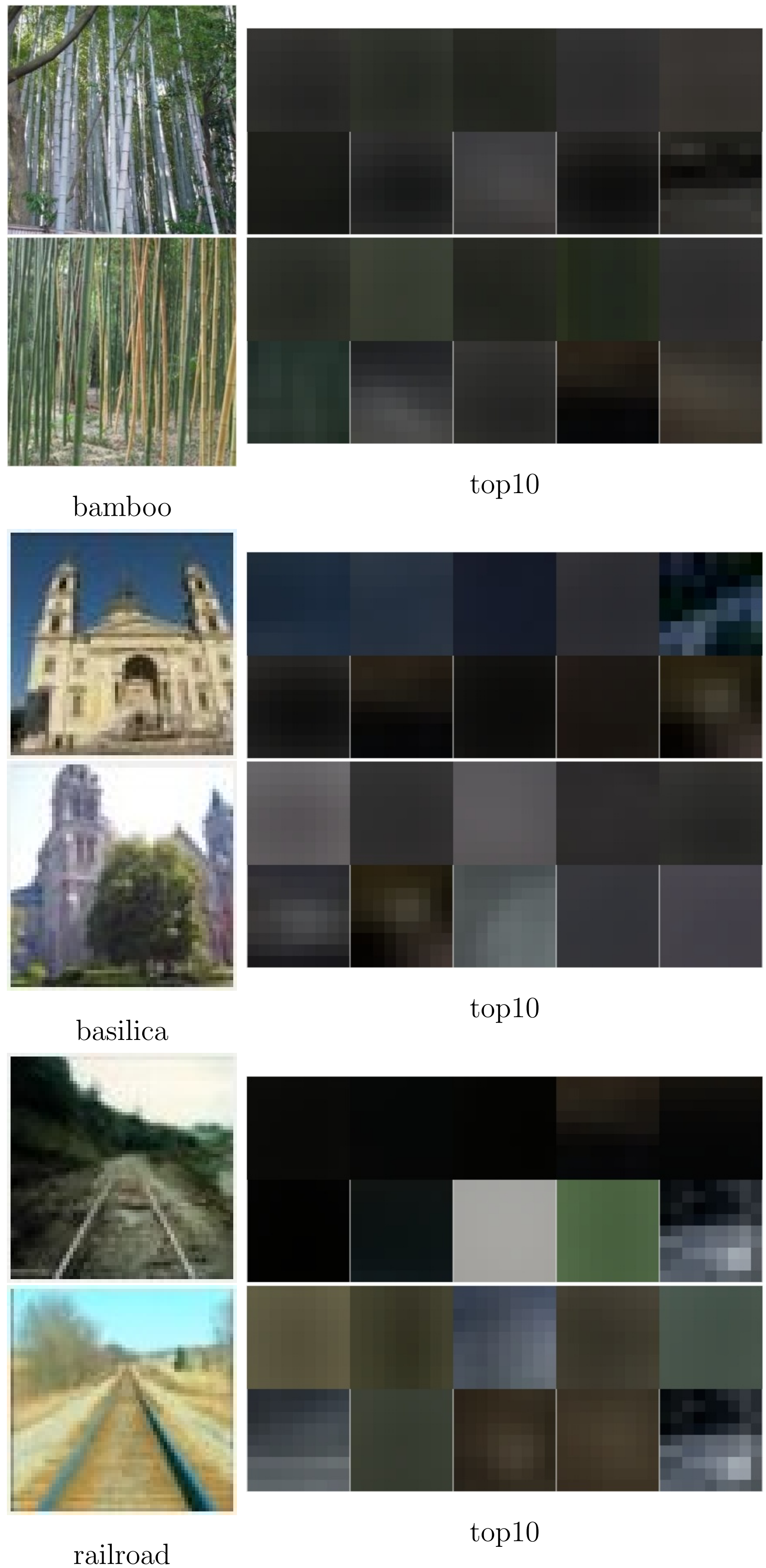


Figure 5: Top 10 words.

Question 6.3   (10 pts, 4-5 lines): Dataset Expansion

1. A way to expand the dataset without using external images is to extract new features from the same dataset. For instance, we can extract interest points features instead of filter-banked features.
2. Flipping will help in bag-of-words with spatial pyramid matching, but not in bag-of-words. Because the former respects spatial information of images, but the latter does not (see **Question 3.2**).

7   Extra Credit: Boosting performance (45 pts)

For the die-hards...

Question 7.1   Better filter bank (10 pts)

The first added is "*Gabor*" filters, they pick up the local spatial and frequency information of images, and sensitive to the edges so that can captures edges of different directions and scales (textures). They are also adaptive to the change of light.

The second added is "*Laplacian*" filters, they pick up the second order derivative of an image isotropically. They are used to detect edges and sharpen the image, but sensitive to noise.

(See Fig. 6.)

Question 7.2   Better image similarity function (5 pts)

I tried *Euclidean* distance and *Cosine* distance for *KNN* classification.

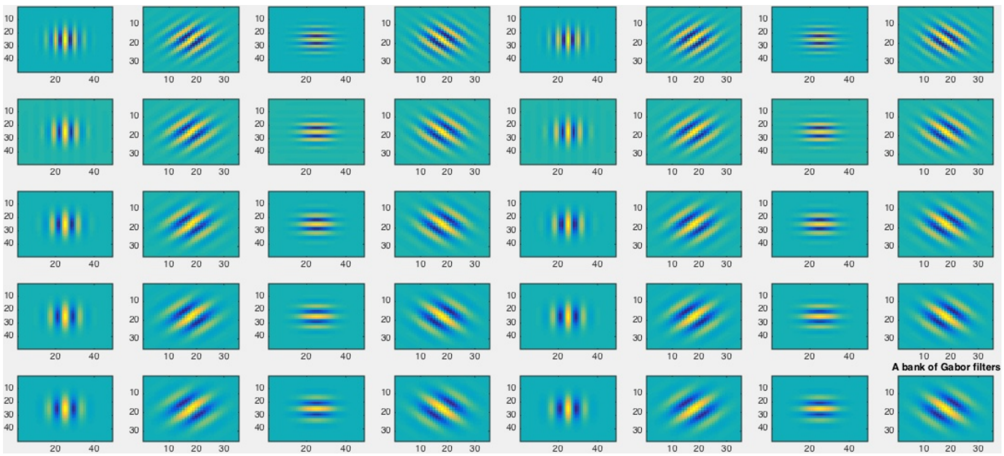
The distance *v.s* accuracy table is shown below in Tab. 2.

Distance	**Similarity**	**Euclidean**	**Cosine**
Accuracy	0.6199	0.4112	0.3988

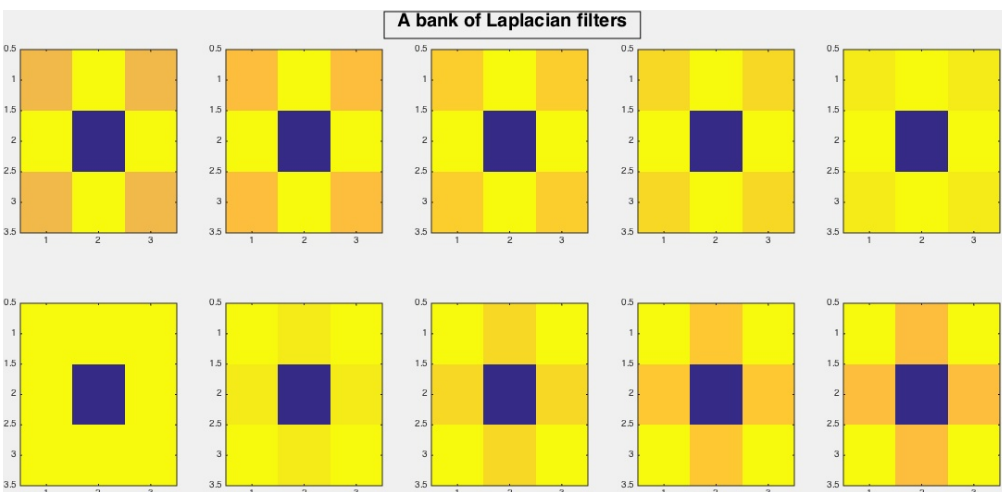
Table 2: **Accuracy versus different distances.** ($k = 11$)

Code for switching from different choice of distances:

```
switch lower(choiceDistance)
    case 'similarity'
        % - similarity
        dist = distanceToSet(testSet, trainSet); % similarity in this case
        [dist_sort, pos]= sort(dist, 'descend'); % sort the distance in descending order
        distance = dist_sort(1:K); % find the top K distances...
        neighbor = trainLabel(pos(1:K)); % and their corresponding labels in trainSet
    case 'euclid'
        % - Euclid distance
        dist = pdist2(testSet.', trainSet.', 'euclid');
        [dist_sort, pos]= sort(dist, 'ascend');
        distance = dist_sort(1:K);
```



Gabor filter



Laplacian filter

Figure 6: Additional filter banks.

```
            neighbor = trainLabel(pos(1:K));
        case 'cosine'
            % - Cosine distance
            dist = pdist2(testSet.', trainSet.', 'cosine');
            [dist_sort, pos]= sort(dist, 'ascend');
            distance = dist_sort(1:K);
            neighbor = trainLabel(pos(1:K));
        case 'kldivergence'
            % - K-L divergence
            KL = @(X, Y)(sum(bsxfun(@times, X, bsxfun(@minus, log2(X),log2(Y)))));
            dist = pdist2(testSet,trainSet, @(testSet,trainSet) KL(testSet,trainSet));
            [dist_sort, pos]= sort(dist, 'ascend');
            distance = dist_sort(1:K);
            neighbor = trainLabel(pos(1:K));
        case 'builtinknn'
            [idx, dist] = knnsearch(trainSet.',testSet.', 'dist','euclid','k',K);
            distance = dist;
            neighbor = trainLabel(idx);
        otherwise
            disp('Undefined distance!\n');
    end
```

Question 7.3 Different Features (15 pts)

I tried using SURF descriptors to get the visual word dictionary and mapping images to word maps. The dictionary size is set to 500.

Then the histograms were got as features for classification *WITHOUT* using SPM.

The accuracy is: *0.3396* (see Fig. 7).

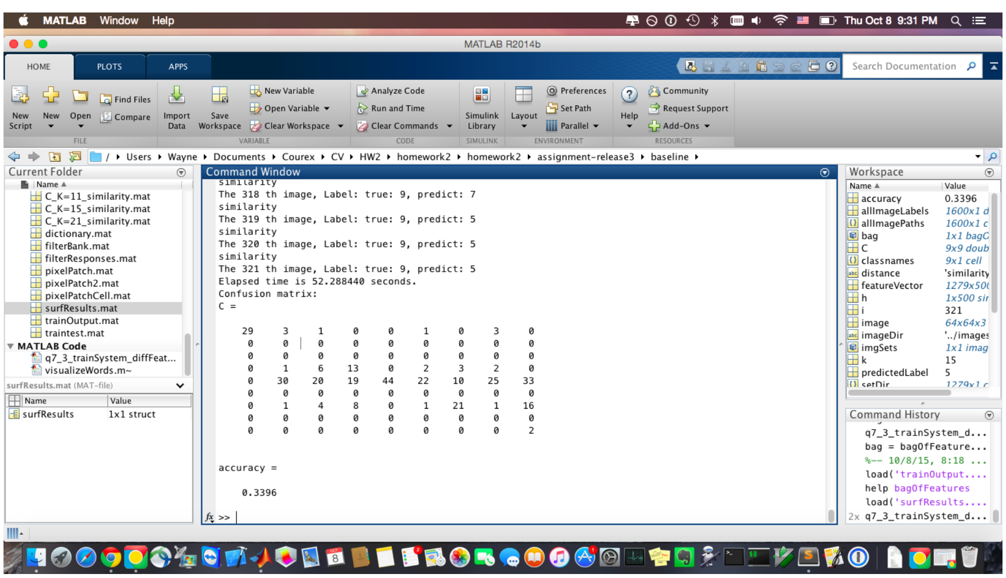


Figure 7: **KNN result using SURF-bag-of-words.** (K=15)

The code is shown below:

```
% This script extracts the visual word dictionary (bag of words) using
% feature descriptors, instead of using filter banks.
%
% Author: WENBO ZHAO (wzhao1@andrew.cmu.edu)
% Date: Oct 8, 2015
% Log: (v0.1)-(first draft, written all the functions)-(Oct 8, 2015)
%      (v0.2)-(modified: fixed bug: improved: )
%
clear all, close all, clc
%% Set directories
load('traintest.mat');
imageDir = '../images'; %where all images are located

setDir  = fullfile(imageDir,trainImagePaths);
imgSets = imageSet(setDir);

%% Get bag of words using Matlab CV toolbox builtin function
% SURF features are used to generate the vocabulary features. Vocabulary is quantized
% using K-means algorithm.
VocabularySize = 500; % dictionary size
PointSelection = 'Grid'; % using SURF feature
tic
bag = bagOfFeatures(imgSets,'VocabularySize', VocabularySize, 'PointSelection', PointSel
toc
%% Map image to dictionary
% can actually use builtin encode, but...
tic;
featureVector = encode(bag,imgSet);
toc;

surfResult.bag = bag;
surfResult.feat = featureVector;
save('surfResult','surfResult');
%% knn classification
% predict
k = 21; % odd number to avoid tie
distance = 'similarity'
% distance = 'euclid'
% distance = 'cosine'
% distance = 'kldivergence'
% distance = 'builtinKnn'
C = zeros(length(unique(trainImageLabels)));
tic
for i = 1:length(testImagePaths)
    image = imread(fullfile(imageDir, testImagePaths{i}));
    h = encode(bag, image);
    predictedLabel = knnClassify(h.', featureVector.', trainImageLabels, k, distance);
    fprintf('The %d th image, Label: true: %d, predict: %d \n', i, testImageLabels(i), p
    C(predictedLabel,testImageLabels(i)) = C(predictedLabel,testImageLabels(i)) + 1;
end
toc
% accuracy
fprintf('Confusion matrix:');
```

```
C
accuracy = trace(C)/sum(C(:))
```

Question 7.4 Encoding (15 pts)

A *'soft-histogram'* is used to encode the features to visual word dictionary[1], in which:

Instead of assigning one feature to a hard one visual word, a membership between (0,1) is used to measure the belongingness to a certain dictionary.

The membership is computed as:

$$U_k = (1/\|X - d_k\|_2^2) / \sum (1/\|X - d_k\|_2^2)$$

code:

```
function [h] = softHist(wordMap, dictionarySize)
% This function uses a 'soft-encoding' method to encode the wordmaps to
% histograms. Instead of assigning one feature to a hard one visual word,
% this function uses a membership between (0,1) to measure the
% belongingness to a certain dictionary.
% The membership is computed as:
%            U_k =  (1/||X-d_k||^2_2)/sum(1/||X-d_k||^2_2)
% - INPUTS: * wordMap: an index map of the size(Image) that maps each pixel
%             response to its closest in the dictionary
%           * dictionarySize: K
% - OUTPUT: * h: a (dictionarySize*1) histogram
%
% Author: WENBO ZHAO (wzhao1@andrew.cmu.edu)
% Date: Oct 8, 2015
% Log: (v0.1)-(first draft, written all the functions)-(Oct 8, 2015)
%      (v0.2)-(modified: fixed bug: improved: )
%
P = pdist2(wordMap(:),[1:1:dictionarySize], 'euclidean');
P = 1./(P.^2);
P = 1./repmat(sum(P,2), [1, dictionarySize]);
h = mean(P,1);
end
```

Question 7.5 Getting Fancy! (0 pts)

I can use SVMs to do classification, but since there is 0 pts...

References

[1] T. Ahonen and M. Pietikäinen, "Soft histograms for local binary patterns," in *Proceedings of the Finnish signal processing symposium, FINSIG*, vol. 5, p. 1, 2007.