Exercise 10 - Image Categorization

Computer Vision

Alberto Montes (malberto@student.ethz.ch) January 17, 2017

Local Feature Extraction

To extract the local features for each image, first it requires to implement the function to extract the grid points positions for each image. The implementation is on Listing 1 where the coordinates of the grid points are found and returned.

```
function vPoints = grid_points(img, nPointsX, nPointsY, border)
%GRID_POINTS
    [h, w] = size(img);
    % Create space of coordinates for each axis
    x_points = linspace(border, w-border, nPointsX);
    y_points = linspace(border, h-border, nPointsY);
    % Mesh the points coordinates
    [X_points, Y_points] = meshgrid(x_points,y_points);
    % Transform the grid into a vector of points
    vPoints = [reshape(Y_points, [nPointsX*nPointsY, 1]),...
        reshape(X_points, [nPointsX*nPointsY, 1])];
    % Round to be used as index
    vPoints = round(vPoints);
end
```

Listing 1: grid_points.m

The next step is for each of the points of the grid at each image, compute the Histogram of oriented Gradients for 4×4 grid pixels. The implementation is in Listing 2 where for each points the descriptor is computed and also the patch of each descriptor is returned.

```
function [descriptors,patches] = descriptors_hog(img,vPoints,cellWidth,
   cellHeight)
%DESCRIPTORS_HOG
    nBins = 8;
    w = cellWidth; % set cell dimensions
    h = cellHeight;
    num = size(vPoints,1);

    descriptors = zeros(num, 4, 4, nBins); % one histogram for each of
   the 16 cells
    patches = zeros(num,4*w*4*h); % image patches stored in rows

    [grad_x, grad_y]=gradient(img);
    orientationGradient = angle(grad_x+1i*grad_y);

    for i = 1:num % for all local feature points
        p = vPoints(i,:);
        p_x_min = p(2)-2*w+1;
        p_x_max = p(2)+2*w;
        p_y_min = p(1)-2*h+1;
        p_y_max = p(1)+2*h;

        patch = img(p_y_min:p_y_max, p_x_min:p_x_max);
```

```
        grad_patch = orientationGradient(p_y_min:p_y_max, p_x_min:p_x_max);
        for ii = 1:4
            for jj = 1:4
                cell = grad_patch(jj:(jj+h),ii:(ii+w));
                [hist, ~] = histcounts(cell, linspace(-pi,pi,nBins+1));
                descriptors(i, ii, jj, :) = hist;
            end
        end
        patches(i,:) = reshape(patch, [1, 4*w*4*h]);
    end % for all local feature points

    descriptors = reshape(descriptors, [num, 4*4*nBins]);
end
```

Listing 2: descriptors_hog.m

Codebook Construction

Once with the descriptors and patches extracted for each image, is time to construct the codebook. To do so, a k-means cluster algorithm is run to obtain an specific number of clusters centers to build the codebook. For the implementation, the codebook size its fixed in 200 codes.

For the k-means algorithm implementation first requires to implement the `findnn` (Listing 3) function which find to each descriptor the other ones which are closest. With this function, the k-means algorithm has been implemented as shown in Listing 4.

```
function [Idx, Dist] = findnn( D1, D2 )
  % input:
  %   D1  : NxD matrix containing N feature vectors of dim. D
  %   D2  : MxD matrix containing M feature vectors of dim. D
  % output:
  %   Idx : N-dim. vector containing for each feature vector in D1
  %         the index of the closest feature vector in D2.
  %   Dist: N-dim. vector containing for each feature vector in D1
  %         the distance to the closest feature vector in D2.


  % Find for each feature vector in D1 the nearest neighbor in D2
  % Compute pairwise distances between each element of D1 and D2
  d = pdist2(D1, D2, 'euclidean');
  % Find the minimum distance and its index
  [Dist, Idx] = min(d, [], 2);
end
```

Listing 3: findnn.m

```
function vCenters = kmeans(vFeatures,k,numIter)

    nPoints  = size(vFeatures, 1);

    % Initialize each cluster center to a different random point.
    cIdx = randsample(1:nPoints, k);
    vCenters = vFeatures(cIdx,:);

    % Repeat for numiter iterations
    for i=1:numIter,
        % Assign each point to the closest cluster
        [clusters, ~] = findnn(vFeatures, vCenters);
```

```

        % Shift each cluster center to the mean of its assigned points
        for j=1:k
            % Get all the points from the 'j' cluster
            clusterPoints = vFeatures(clusters==j,:);
            % Recompute the cluster center
            vCenters(j,:) = mean(clusterPoints, 1);
        end

        disp(strcat(num2str(i),'/',num2str(numIter),' iterations completed.'));
    end;

end
```

Listing 4: `kmeans.m`

Finally the whole pipeline of creating the codebook is coded in Listing 5. The pipeline iterates over all the images and for each on first convert it to gray scale, then obtain the grid points coordinates with the `grid_points` function. Then obtain the descriptor for each of this points and the image patch using the `descriptors_hog` function. Finally with all the image's descriptors, cluster them with the `kmeans` algorithm and obtain the cluster centers as classification codebook. The whole pipeline implementation is on Listing 5.

```
function vCenters = create_codebook(nameDir,k,numiter)

    cellWidth = 4;
    cellHeight = 4;
    nPointsX = 10;
    nPointsY = 10;
    nPoints = nPointsX * nPointsY;
    border = 8;

    vImgNames = dir(fullfile(nameDir,'*.png'));

    nImgs = length(vImgNames);
    nDescriptors = nPoints * nImgs;
    vFeatures = zeros(nDescriptors, 128); % 16 histograms containing 8 bins
    vPatches = zeros(nDescriptors, 16*16); % 16*16 image patches

    % Extract features for all images
    c = 1;
    for i=1:nImgs,

        disp(strcat('  Processing image ', num2str(i),'...'));

        % Load the image
        img = double(rgb2gray(imread(fullfile(nameDir,vImgNames(i).name))));

        % Collect local feature points for each image
        vPoints = grid_points(img, nPointsX, nPointsY, border);
        % Compute a descriptor for each local feature point
        [descriptors, patches] = descriptors_hog(img, vPoints, cellWidth, cellHeight);
        % Create hog descriptors and patches
        vFeatures(((c-1)*nPoints+1):c*nPoints,:) = descriptors;
        vPatches(((c-1)*nPoints+1):c*nPoints,:) = patches;

```

```
        c = c + 1;

    end;
    disp(strcat('    Number of extracted features:',num2str(size(
vFeatures,1))));

    % Cluster the features using K-Means
    disp(strcat('  Clustering...'));
    vCenters = kmeans(vFeatures, k, numiter);

    % Visualize the code book
    disp('Visualizing the codebook...');
    visualize_codebook(vCenters,vFeatures,vPatches,cellWidth,cellHeight)
;
    disp('Press any key to continue...');
    pause;

end
```

Listing 5: `create_codebook.m`

For the training data given with the assignment, and computing a codebook with size 200, the visualization of the codebook is on Figure 1 where the patches of the closest descriptors to the codebook cluster's centers are plot.

Bag-of-Words Image Representation

Once the codebook has been defined, for each image is extracted a bunch of descriptors which each one will be match to one of the codebook's code or visual word. The purpose of the bag-of-words is to represent an image as an histogram of the visual words appearing on it, and with this histogram then perform a classification. To do so, first the `bow_histogram` function has been implemented (in Listing 6) which for each image given extract the histogram of visual words, given the codebook previously found.

```
function histo = bow_histogram(vFeatures, vCenters)
    %BOW_HISTOGRAM
    % input:
    %   vFeatures: MxD matrix containing M feature vectors of dim. D
    %   vCenters : NxD matrix containing N cluster centers of dim. D
    % output:
    %   histo    : N-dim. vector containing the resulting BoW
    %              activation histogram.


    % Match all features to the codebook and record the activated
    % codebook entries in the activation histogram "histo".
    N = size(vCenters, 1);
    [~, words] = findnn(vFeatures, vCenters);
    histo = histcounts(words, 1:(N+1));

end
```

Listing 6: `bow_histogram.m`

To perform a classification, is necessary to precompute this BoW histograms to all the training examples and store to further classify the test dataset. This is performed in the function `create_bow_histogram` on Listing 7.

```
function vBoW = create_bow_histograms(nameDir, vCenters)

    vImgNames = dir(fullfile(nameDir,'*.png'));
```

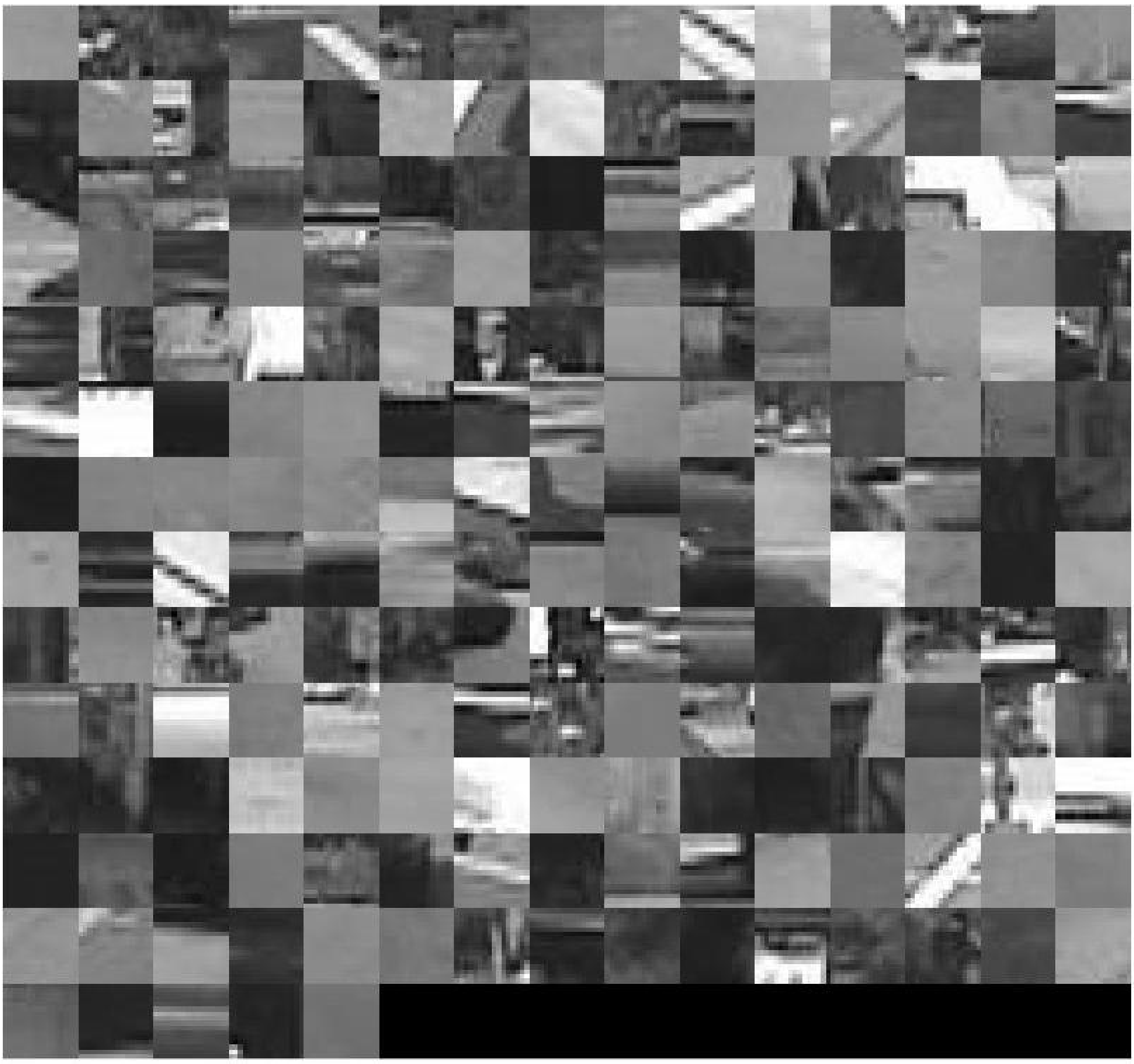


Figure 1: Visualization of the codebook for $K = 200$.

```
nImgs = length(vImgNames);
nWords = size(vCenters, 1);
vBoW   = zeros(nImgs, nWords);

cellWidth = 4;
cellHeight = 4;
nPointsX = 10;
nPointsY = 10;
border = 8;

% Extract features for all images in the given directory
for i=1:nImgs,
    disp(strcat('  Processing image ', num2str(i),'...'));

    % Load the image
    img = double(rgb2gray(imread(fullfile(nameDir,vImgNames(i).name)
)));

    % Collect local feature points for each image
```

```
        vPoints = grid_points(img, nPointsX, nPointsY, border);
        % Compute a descriptor for each local feature point
        [vFeatures, ~] = descriptors_hog(img, vPoints, cellWidth,
    cellHeight);

        % Create a BoW activation histogram for this image
        vBoW(i,:) = bow_histogram(vFeatures, vCenters);


    end;

end
```

Listing 7: `create_bow_histograms.m`

Nearest Neighbor Classification

With all the Bag of Words computed for the training samples and also for the testing samples, two methods to classify whether in an image there is a car or not are proposed. The first method consist on the Nearest Neighbor where for each test samples is computed the distance of the BoW to each of the training samples and the class of the closest distance is returned. The implementation is on Listing 8 which it has been very easy to code.

```
function sLabel = bow_recognition_nearest(histogram,vBoWPos,vBoWNeg)
    %BOW_RECOGNITION_NEAREST
    % Find the nearest neighbor in the positive and negative sets
    % and decide based on this neighbor

    % Compute the distances to the Positive and Negative samples
    dPos = pdist2(histogram, vBoWPos);
    dNeg = pdist2(histogram, vBoWNeg);

    % See the shortest distance to each of the samples
    distPos = min(dPos);
    distNeg = min(dNeg);

    % Return the label depending of the shortest distance
    if (distPos < distNeg)
        sLabel = 1;
    else
        sLabel = 0;
    end

end
```

Listing 8: `bow_recognition_nearest.m`

The result of the implementation with the default value of codebook size is a 76.76% accuracy detecting cars.

Bayesian Classification

The second classification method proposed consist on a Bayesian Classification. On this method, for each of the features of each image, the prior distributions is found considering that it follows a normal distribution. So with the training data the $P(hist|car)$ and $P(hist|!car)$ are computed and considering that $P(car) = P(!car) = 0.5$, to classify each image, only is necessary to evaluate $log[P(hist|car)]$ and $log[P(hist|!car)]$. The logarithmic is used for numerical stability, so the implementation is listed in Listing 9.

```
function label = bow_recognition_bayes( histogram , vBoWPos , vBoWNeg )


    [muPos , sigmaPos] = computeMeanStd(vBoWPos);
    [muNeg , sigmaNeg] = computeMeanStd(vBoWNeg);

    % Calculating the probability of appearance each word in observed
    % histogram according to normal distribution in each of the positive
    % and negative bag of words.

    P_hist_pos_j = normpdf(histogram , muPos , sigmaPos);
    P_hist_neg_j = normpdf(histogram , muNeg , sigmaNeg);
    % Check if there is any NaN at the generated probability
    P_hist_pos_j(isnan(P_hist_pos_j)) = 1;
    P_hist_neg_j(isnan(P_hist_neg_j)) = 1;
    % Compute the join probability
    P_hist_pos = sum(log(P_hist_pos_j));
    P_hist_neg = sum(log(P_hist_neg_j));

    if P_hist_pos > P_hist_neg
        label = 1;
    else
        label = 0;
    end


end
```

Listing 9: `bow_recognition_bayes.m`

The result of the implementation with the default value of codebook size is a 70.70% accuracy detecting cars.

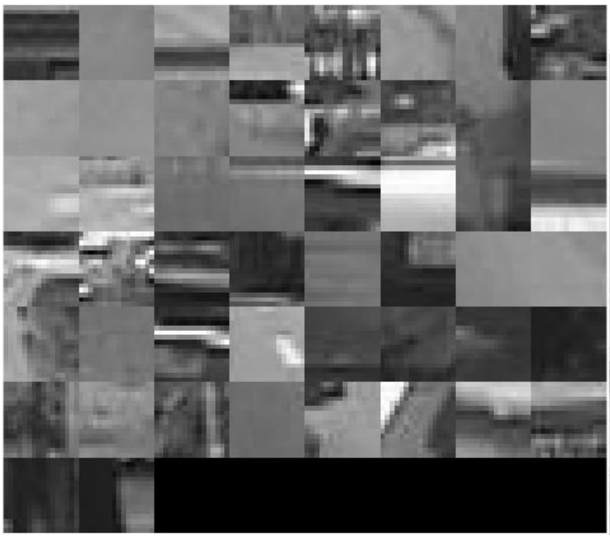
Results and Conclusions

To go further, the classification has been performed for different codebook sizes, going from 10, to 300. Between 20 and 50 has been observed the best results using Bayes and Nearest Neighbors classifiers achieving more than 95% of accuracy. As the codebook size increases, the performance of both classifiers decreases due to the fact that as bigger the codebook, more combinations of BoW and with an small dataset (only 100 training instances) the precision at the time to classify decreases.

In the Figure 2 there are the codebooks of the best codebook sizes for Nearest Neighbor classification and Bayesian classification respectively.

Size Codebook	**Nearest Neighbor**	**Bayesian**
10	50.50%	50.50%
20	93.94%	**98.99%**
50	**96.97%**	90.90%
100	86.86%	83.83%
150	81.81%	81.81%
200	76.76%	70.70%
250	64.64%	66.66%
300	65.65%	62.62%

Table 1: Results for different codebook sizes.


(a) $K = 50$

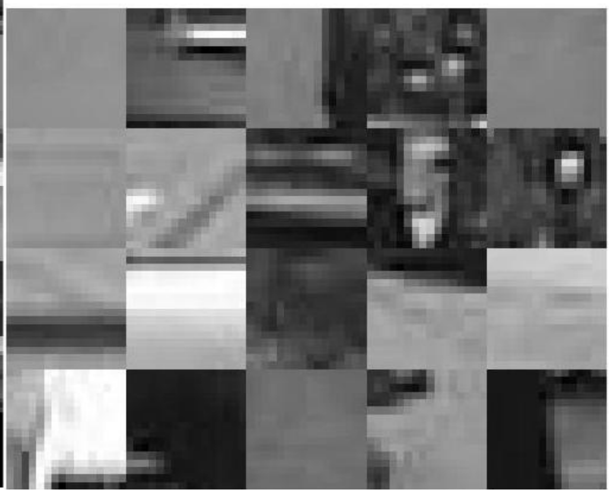

(b) $K = 20$

Figure 2: Visualize codebook with better classification accuracy.