Exercise 10 - Image Categorization

Computer Vision

Alberto Montes (malberto@student.ethz.ch)   December 20, 2016

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.

```
1  function vPoints = grid_points(img, nPointsX, nPointsY, border)
2  %GRID_POINTS
3      [h, w] = size(img);
4      % Create space of coordinates for each axis
5      x_points = linspace(border, w-border, nPointsX);
6      y_points = linspace(border, h-border, nPointsY);
7      % Mesh the points coordinates
8      [X_points, Y_points] = meshgrid(x_points,y_points);
9      % Transform the grid into a vector of points
10     vPoints = [reshape(Y_points, [nPointsX*nPointsY, 1]),...
11         reshape(X_points, [nPointsX*nPointsY, 1])];
12     % Round to be used as index
13     vPoints = round(vPoints);
14 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.

```
1  function [descriptors,patches] = descriptors_hog(img,vPoints,cellWidth,
       cellHeight)
2  %DESCRIPTORS_HOG
3      nBins = 8;
4      w = cellWidth; % set cell dimensions
5      h = cellHeight;
6      num = size(vPoints,1);
7
8      descriptors = zeros(num, 4, 4, nBins); % one histogram for each of
       the 16 cells
9      patches = zeros(num,4*w*4*h); % image patches stored in rows
10
11     [grad_x, grad_y]=gradient(img);
12     orientationGradient = angle(grad_x+1i*grad_y);
13
14     for i = 1:num % for all local feature points
15         p = vPoints(i,:);
16         p_x_min = p(2)-2*w+1;
17         p_x_max = p(2)+2*w;
18         p_y_min = p(1)-2*h+1;
19         p_y_max = p(1)+2*h;
20
21         patch = img(p_y_min:p_y_max, p_x_min:p_x_max);
```

```
22        grad_patch = orientationGradient(p_y_min:p_y_max, p_x_min:
    p_x_max);
23        for ii = 1:4
24            for jj = 1:4
25                cell = grad_patch(jj:(jj+h),ii:(ii+w));
26                [hist, ~] = histcounts(cell, linspace(-pi,pi,nBins+1));
27                descriptors(i, ii, jj, :) = hist;
28            end
29        end
30        patches(i,:) = reshape(patch, [1, 4*w*4*h]);
31    end % for all local feature points
32
33    descriptors = reshape(descriptors, [num, 4*4*nBins]);
34 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.

```
1 function [Idx, Dist] = findnn( D1, D2 )
2   % input:
3   %   D1  : NxD matrix containing N feature vectors of dim. D
4   %   D2  : MxD matrix containing M feature vectors of dim. D
5   % output:
6   %   Idx : N-dim. vector containing for each feature vector in D1
7   %         the index of the closest feature vector in D2.
8   %   Dist: N-dim. vector containing for each feature vector in D1
9   %         the distance to the closest feature vector in D2.
10
11
12   % Find for each feature vector in D1 the nearest neighbor in D2
13   % Compute pairwise distances between each element of D1 and D2
14   d = pdist2(D1, D2, 'euclidean');
15   % Find the minimum distance and its index
16   [Dist, Idx] = min(d, [], 2);
17 end
```

Listing 3: `findnn.m`

```
1 function vCenters = kmeans(vFeatures,k,numiter)
2
3     nPoints  = size(vFeatures, 1);
4
5     % Initialize each cluster center to a different random point.
6     cIdx = randsample(1:nPoints, k);
7     vCenters = vFeatures(cIdx,:);
8
9     % Repeat for numiter iterations
10    for i=1:numiter,
11        % Assign each point to the closest cluster
12        [~, clusters] = findnn(vFeatures, vCenters);
```

```
14          % Shift each cluster center to the mean of its assigned points
15          for j=1:k
16              % Get all the points from the 'j' cluster
17              clusterPoints = vFeatures(clusters==j,:);
18              % Recompute the cluster center
19              vCenters(j,:) = mean(clusterPoints, 1);
20          end
21
22          disp(strcat(num2str(i),'/',num2str(numiter),' iterations
    completed.'));
23      end;
24
25
26 end
```

<div align="center">Listing 4: <code>kmeans.m</code></div>

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.

```
1 function vCenters = create_codebook(nameDir,k,numiter)
2
3      cellWidth = 4;
4      cellHeight = 4;
5      nPointsX = 10;
6      nPointsY = 10;
7      nPoints = nPointsX * nPointsY;
8      border = 8;
9
10     vImgNames = dir(fullfile(nameDir,'*.png'));
11
12     nImgs = length(vImgNames);
13     nDescriptors = nPoints * nImgs;
14     vFeatures = zeros(nDescriptors, 128); % 16 histograms containing 8
    bins
15     vPatches = zeros(nDescriptors, 16*16); % 16*16 image patches
16
17     % Extract features for all images
18     c = 1;
19     for i=1:nImgs,
20
21         disp(strcat(' Processing image ', num2str(i),'...'));
22
23         % Load the image
24         img = double(rgb2gray(imread(fullfile(nameDir,vImgNames(i).name)
    )));
25
26         % Collect local feature points for each image
27         vPoints = grid_points(img, nPointsX, nPointsY, border);
28         % Compute a descriptor for each local feature point
29         [descriptors, patches] = descriptors_hog(img, vPoints, cellWidth
    , cellHeight);
30         % Create hog descriptors and patches
31         vFeatures(((c-1)*nPoints+1):c*nPoints,:) = descriptors;
32         vPatches(((c-1)*nPoints+1):c*nPoints,:) = patches;
```

```
33
34          c = c + 1;
35
36      end;
37      disp(strcat('    Number of extracted features:',num2str(size(
    vFeatures,1))));
38
39      % Cluster the features using K-Means
40      disp(strcat('  Clustering...'));
41      vCenters = kmeans(vFeatures, k, numiter);
42
43      % Visualize the code book
44      disp('Visualizing the codebook...');
45      visualize_codebook(vCenters,vFeatures,vPatches,cellWidth,cellHeight)
    ;
46      disp('Press any key to continue...');
47      pause;
48
49 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.

```
1 function histo = bow_histogram(vFeatures, vCenters)
2      %BOW_HISTOGRAM
3      % input:
4      %   vFeatures: MxD matrix containing M feature vectors of dim. D
5      %   vCenters : NxD matrix containing N cluster centers of dim. D
6      % output:
7      %   histo    : N-dim. vector containing the resulting BoW
8      %              activation histogram.
9
10
11      % Match all features to the codebook and record the activated
12      % codebook entries in the activation histogram "histo".
13      N = size(vCenters, 1);
14      [~, words] = findnn(vFeatures, vCenters);
15      histo = histcounts(words, 1:(N+1));
16
17 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.

```
1 function vBoW = create_bow_histograms(nameDir, vCenters)
2
```

```
3    vImgNames = dir(fullfile(nameDir,'*.png'));
4    nImgs = length(vImgNames);
5    nWords = size(vCenters, 1);
6    vBoW  = zeros(nImgs, nWords);
7
8    cellWidth = 4;
9    cellHeight = 4;
10   nPointsX = 10;
11   nPointsY = 10;
12   border = 8;
13
14   % Extract features for all images in the given directory
15   for i=1:nImgs,
16       disp(strcat('  Processing image ', num2str(i),'...'));
17
18       % Load the image
19       img = double(rgb2gray(imread(fullfile(nameDir,vImgNames(i).name)
     )));
20
21       % Collect local feature points for each image
22       vPoints = grid_points(img, nPointsX, nPointsY, border);
23       % Compute a descriptor for each local feature point
24       [vFeatures, ~] = descriptors_hog(img, vPoints, cellWidth,
     cellHeight);
25
26       % Create a BoW activation histogram for this image
27       vBoW(i,:) = bow_histogram(vFeatures, vCenters);
28
29
30   end;
31
32 end
```

Listing 7: create_bow_histograms.m

Nearest Neighbor Classification

Bayesian Classification

Create a table that for different values of K (sizeCodebook) the result for each of the classification methods.

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