Prof. Bernt Schiele, Dr. Mario Fritz
<{schiele,mfritz}@mpi-inf.mpg.de>

Seon Joon Oh, Alina Dima, Rakshith Shetty
<{joon, aldima, rshetty}@mpi-inf-mpg.de>

Exercise 2: Local descriptors for Retrival and Classification

In this exercise you will utilize local SIFT descriptors to solve two problems, image retrieval and classification. Questions 1-3 focus on developing a simplified image retrieval system. In order to retrieve fast a number of initial candidates, you will use visual words and a vocabulary tree. The vocabulary tree is obtained by hierarchically clustering image descriptors from a set of training images.

In question 4 you will build a binary classifier which classifies if an image contains a dog or an airplane. Here again, you will use SIFT descriptors and extract bag-of-words features for each image and use these features to train a SVM classifier.

Download the file `exercise2.zip` from the course web page. The file contains two datasets: the *Covers_test* and *Covers_train* image folders which is used for the image retrieval model and the *st10_test* and *st10_train* image folders which is used for the classification problem.

Covers data: The *Covers* train and test images correspond to the same set of objects, namely CD covers, photographed from different viewpoints. The images are organized so that ordering them by name provides the right correspondences. So 1st image in test (ukbench00989.jpg) corresponds to 1st image in train (ukbench00988.jpg). The train data will be used as the database on which you will build the vocabulary tree. Test data is used as queries for which you will retrieve matching images from the train set.

St10 data: The *st10* train and test images contain images with 2 object classes, "dogs" and "planes". The files are named according to the object classes. The train set contains 400 images (200 each class) and test set contains 100 images. You will use the training data to extract vocabulary and train SVM classifier and test data to measure the performance of the classifier.

The provided package also contains a set of Matlab functions some of which are missing the implementation and only define interface with the rest of the code. Your task will be to implement the missing parts, run the system with different parameters, and comment on the results. You can use the provided script `ex2.m` to see in which context your code will be executed. This script will also be used to test your solution.

Here we use the DoG feature point detector and the SIFT descriptor implementation from the VLFeat [1] library. This library also provides an implementation of the hierarchical and regular k-means clustering algorithm, which we will use to obtain the vocabulary tree representation. The library is included in the archive file.

Question 1: Learning a vocabulary tree of visual words (10 points)

a) Write a function `Create_vocabulary_tree.m` which takes as input the name of a directory and outputs a variable `vocabulary_tree`, containing the learnt visual words and the vocabulary tree.

   The function should do the following:

   - load each image in the input directory[2];
   - compute interest points and extract SIFT feature descriptors around each interest point;
   - cluster the features extracted from all images with the hierarchical k-means algorithm.

   In your implementation, both the feature point detector and descriptor will be provided by the function `vl_sift` from the VLFeat library:

```
img= single ( rgb2gray( I ) );
[sift_frames,cur_sift_desc] = vl_sift(img);
```

   `vl_sift` detects feature points using DoG and computes SIFT descriptors given a single-precision, gray-scale image. The variable `sift_frames` will contain a matrix with the number of columns equal to the number of detected interest points, in which each column contains a location vector `[x_coord; y_coord; scale; orientation]`. `cur_sift_desc` will contain a matrix with the number of columns equal to the number of detected interest points, in which each column contains the 128 dimensional SIFT descriptor.

[1] `www.vlfeat.org`
[2] a list of all images in the directory given by `vocab_dir` can be obtained with the command: `vImgNames = dir([vocab_dir '/*.jpg']);`

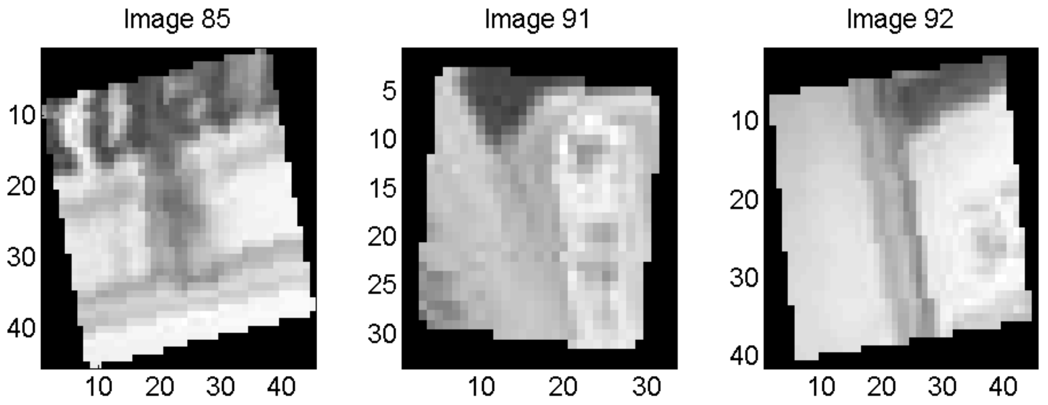


Figure 1: Rotated image patches of features represented by the same visual word

Compute *all* SIFT descriptors from *all* images in the folder into a matrix named `all_sift_desc`, with the number of columns equal to the number of all computed SIFT descriptors, and each column containing a SIFT descriptor.

Finally compute the visual words and the vocabulary tree using the function `vl_hikmeans`:

```
    [vocabulary_tree] = vl_hikmeans(all_sift_desc, branching_factor, min_vocabulary_size);
```

`vocabulary_tree` is a recursive structure. Each node of the structure contains a variable `centers` with the columns equal to the cluster centroids (the visual words) at that level of the tree. Additionally each node contains an array of children structures `sub`, with the same recursive structure as their parent node. Both the number of centroids and the children equal in general the `branching_factor`: the number of branches/children generating from each node. Only the root node contains some additional parameter variables: `K` (=`branching_factor`) and `depth`. The latter defines implicitly the vocabulary size: $\texttt{branching\_factor}^{\texttt{depth}}$. The vocabulary size equals the smallest power of `branching_factor` which is larger or equal to `min_vocabulary_size`.

In your experiment you may specify `branching_factor`=8 and `min_vocabulary_size`=4096, which corresponds to `depth`=4.

b) Build up the vocabulary tree from the training images in the folder *Covers_train*.

c) Familiarize yourself with the vocabulary tree now.

The variable `vocabulary_tree` is used to associate a SIFT descriptor to a visual word, namely a leaf of the tree. A queried SIFT descriptor is compared to the cluster centers of the root node of the tree and passed to the one child node, the cluster center of which is closest to the queried SIFT descriptor. The process proceeds until the queried descriptor is passed to a leaf node, i.e. the visual word that it correspond to.

For passing a feature descriptor through the tree you will use the function `vl_hikmeans` from VLFeat:

```
    path_to_leaf = vl_hikmeanspush(vocabulary_tree,one_sift_descriptor);
```

The output is the variable `path_to_leaf`, a vector array specifying the visual word by its path from the root to the leaf.

As an example, in case of `vocabulary_tree.depth`=4, a leaf is specified by the centroid of

```
    vocabulary_tree.sub(path_to_leaf(1)).sub(path_to_leaf(2))....
       sub(path_to_leaf(3)).centers(path_to_leaf(4))
```

d) Implement a function to count the vocabulary words, i.e. the number of cluster centroids at the leaves of the vocabulary tree. Write a function which traverses the tree `vocabulary_tree` up to the leaves and counts the `centers`, i.e. the cluster centroids, assigned at the leaf node.

Note that a tree node is a leaf when the children structure `sub` counts zero elements (`numel(parent.sub)<1`).

e) Implement a function `Show_visual_word` which computes again all SIFT descriptor from all images in the *Covers_train* folder, passes the feature descriptors through the vocabulary tree, and shows the feature patches which are represented by the same, specified, visual word.

Generate a random path to leaf and run the function with it:

```
random_path=randi(vocabulary_tree.K,[1,vocabulary_tree.depth]);
Show_visual_word(random_path,'./Covers_train',vocabulary_tree)
```

In the function, a `path_to_leaf` vector equal to `random_path` means a matching visual word. For the first few matching visual words, take the corresponding feature point location vector `sift_frames` and use it to extract the corresponding patch from the image. In particular, for each matching visual word cut from the image a patch of size 6 x `scale`, centered at the coordinates `[x_coord; y_coord]` (you do not need to rotate the patch by the `orientation` angle, you may just print this value in the title of the image).

Can you see the visual similarity of the leaf words?

For debugging purposes, we have included into this zip a saved `vocabulary_tree` variable (you can retrieve it by typing `load vocabulary_tree_variable`). The illustrated feature patches in figure 1 are obtained for `random_path=[7 6 6 2]`.

Question 2: Inverted file index generation (10 points)

For fast retrieving the image matches, it is not just necessary to generate a codebook. The inverted file index associates to each visual word a number of candidate image matches and a corresponding score. Given a new image, the vocabulary tree provides therefore the visual words, and the inverted file index is used to retrieve image candidates with each of them.

a) Write a function `Invert_file_index` which takes as input a folder name and a learned vocabulary tree, and outputs the variable `ifindex`.

The function should do the following:

- load each image in this directory;
- compute interest points and extract SIFT descriptors around each interest point;
- pass all descriptors through the vocabulary tree down to the leaves;
- record for each leaf which images had features represented by that visual word, and how many of those visual words each image had.

This record is the inverted file index, the variable `ifindex`. You will implement it with a vector array. The size of this vector array will be the potential number of leaves of the vocabulary tree $\texttt{branching\_factor}^{\texttt{depth}}$. Each element `i` of the vector will be a structure with two fields: `ifindex(i).images` and `ifindex(i).scores`. `ifindex(i).images` will be a list of images which had feature descriptors represented by the corresponding visual word at leaf `i`. `ifindex(i).scores` will list how many descriptors from the corresponding images were represented by the visual word `i`. As an example, `ifindex(i).scores(j)==3` means that image `ifindex(i).images(j)` had 3 visual words `i`. You will initialize the vector `ifindex` in the following way:

```
nleaves=vocabulary_tree.K^vocabulary_tree.depth;
ifindex(nleaves).images=[];
ifindex(nleaves).scores=[];
```

For passing a descriptor through a tree from the root to the leaves you will use the VLFeat function `vl_hikmeans`. The function, as described above, returns `path_to_leaf`.

In order to provide correspondence between the vector index `i` of `ifindex` and `path_to_leaf`, two function have been provided:

```
index=Path2index(path_to_leaf,vocabulary_tree.K);
path_to_leaf=Index2path(index,vocabulary_tree.K,vocabulary_tree.depth);
```

b) Build up the inverted file from the training images in the folder *Covers_train.*

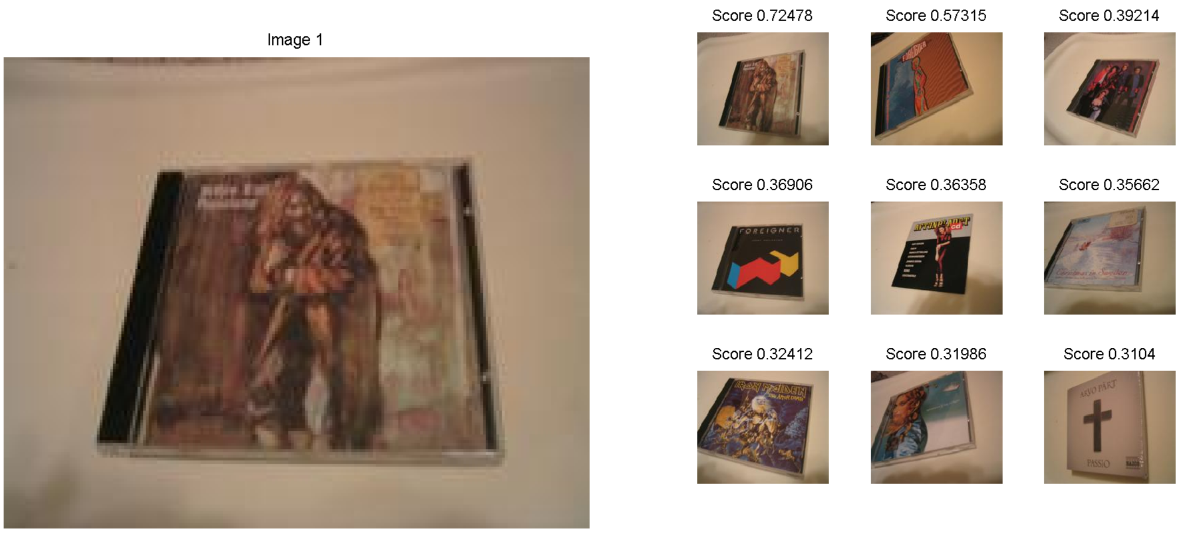
c) Normalize the inverted file index with the *Term Frequency - Inverted Document Frequency* (tf-idf).

It is common practice to normalize the scores of the inverted file index according to the number of visual words in each image and, more generally, to the statistics of the considered dataset, e.g. how many images contain a particular visual word.

You will normalize each score `ifindex(i).scores(j)` according to tf-idf:

$$\texttt{ifindex(i).scores(j)} = \frac{n_{di}}{n_d} \log \frac{N}{N_i} \tag{1}$$

where N is the total number of images, N_i the number of images word i occurs in, in whole database, n_{di} the number of occurrences of word i in image d (where d is `ifindex(i).images(j)`), n_d the number of visual words in image d.



(a) (b)

Figure 2: Query image (a) and retrieved images (b) ordered according to their score

Question 3: Fast image retrieval and scoring

Now that both the vocabulary tree and the inverted index have been learnt, we can fast retrieve images from the test dataset, corresponding to a query image. If properly implemented, this image retrieval engine can scale up to large datasets. The best candidates are verified with an accurate geometric test, based on the Homography estimation, having assumed that the images represent planar objects.

a) Write a function `Retrieve_best_candidates` which, given an input image, computes the best matching candidate images by using the vocabulary tree and the inverted file index.

   The function should do the following:

   - compute interest points and extract SIFT features around each interest point;
   - pass all descriptors through the vocabulary tree down to the leaves;
   - use the index associated with each leaf to cast votes on the images at that inverted file index according to the corresponding scores
   - accumulate the scores for all images
   - order the candidate images according to their scores and return the best 10

   For each descriptor, you will compute `path_to_leaf` with `vl_hikmeanspush`. Then you will compute `index` with *Path2index*. You will use `index` to retrieve the images containing the corresponding visual word, i.e. `ifindex(index).images`. You will add each of this images to a vector `all_voted_images`, and you will increment a score, in the corresponding vector `all_voting_scores`. As an example, if image `ifindex(index).images(j)` was added in position `pos` to vector `all_voted_images` (`all_voted_images(pos)==ifindex(index).images(j)`), we would increment the corresponding score by:

   ```
   all_voting_scores(pos)=all_voting_scores(pos)+ifindex(index).scores(j);
   ```

   You will order the candidate images by using the `sort` Matlab command on `all_voting_scores`

b) Test the retrieval function on various images from the test set.

c) A test image is considered retrieved if the best scoring candidate is the correctly corresponding image.

   You will use the folder listing order to assess a correct candidate: using `dir` provides filenames ordered by their name. True correspondences have the same listing order position in *Covers_test* and *Covers_train*.

   Retrieve the best candidate for each image in the test folder *Covers_test*. Use the counting variable `retriev_accuracy` to count the number of correctly retrieved test images using the vocabulary tree and inverted file index (as from the output of function `Retrieve_best_candidates`). The final retrieval scores will be given by the percentage of correctly retrieved images on the total number of processed images.

   Report performance of the image retrieval algorithm in term of percentage of correctly retrieved images.

To ease the debug of your code, figure 2 shows the output of function `Retrieve_best_candidates` for the first image in the test folder, both the ranked matched images and the matching score, after loading `vocabulary_tree_variable.mat`, computing the inverted file index and normalizing it according to tf-idf.

Question 4: Classification using local features (10 points)

Next we will build image classification system to distinguish if an image contains a dog or an airplane. First we will encode the images using bag-of-words features based on a vocabulary of local SIFT descriptors. Then we train a Support Vector Machine based binary classifier which takes the image feature as input.

a) Write a function `Create_codebook` which takes as input an image folder and creates a codebook from local descriptors. The function should do the following:
   - load each image in this directory.
   - compute interest points and extract SIFT descriptors around each interest point
   - collect all descriptors from all images and cluster them using k-means to obtain the cluster centers. This acts as the codebook.

```
[codebook, ~] = vl_kmeans(single(all_sift_desc), vocab_size);
```

b) Write a function `extract_bow_features` which takes as input an image folder and the codebook and extracts bow features for each image. The function should do the following:
   - load each image in this directory.
   - compute interest points and extract SIFT descriptors around each interest point (use `vl_sift`).
   - Assign each descriptor to the closest entry in the codebook. Use euclidean distance to measure the distance. Count the number of descriptors assigned to each codebook entry to obtain a histogram of `vocab_size` dimensions. This histogram is the bag-of-words representation of the image. Repeat for all images.
   - This function also returns the label correspoding to each image, code for which is already present. Labels are needed to train and test the SVM.

c) Using the `extract_bow_features` fucntion, extract bow features for both training and test images.

d) Train a SVM classifier to classify the images using the training set. Use the `fitcsvm` matlab function to train the SVM model. Familiarize yourself with different parameters of the SVM. For e.g. you can change the kernel used with parameter `'KernelFunction', 'polynomial'`.

```
SVMmodel = fitcsvm(features, label, 'Standardize', true);
```

e) Use the trained model to make predictions on the training and test set. Use the matlab function `predict`. Compute classification accuracy on both training and test sets. Compare the training and test accuracies for atleast. 'linear', 'polynomial' and 'RBF' kernels. Analyze the results.

Further reading

The retrieval algorithm implemented is a simplification of:

D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2161-2168, June 2006

On a larger database of CD covers (50000 images), the authors demonstrated a retrieval performance of less than a second.

Please turn in your solution by sending an email to Rakshith Shetty `<rshetty@mpi-inf.mpg.de>` *including all relevant in a m-files before Tuesday, May 23rd, 23:59. in a single .zip or tar.gz file*