ETH Zürich
Computer Science Department

Prof. Marc Pollefeys
Prof. Siyu Tang
Prof. Vittorio Ferrari
HS 20

Computer Vision - Exercise 10
Image Categorization

Hand-out: 10-12-2020
Hand-in: 19-12-2020
denys.rozumnyi@inf.ethz.ch

Recognizing objects in images is a fundamental and very challenging computer vision problem. The bag-of-words framework, where images are represented by histograms over visual words, is currently one of the most successful approaches to object recognition. In this exercise, we will step-by-step implement a bag-of-words image classifier that decides whether a test image contains a car (back view) or not. The subsequent stages of building the bag-of-words image classifier are the following:

- feature detection
- feature description
- codebook construction
- bag-of-words image representation
- image classification

The following tasks will lead you through the different steps of the training and recognition procedures for this popular classification pipeline.
We provide the full pipeline of the entire exercise in `bow_evaluation`. Also we provide template code, as well as training and test images for this exercise. Your task is to complete missing parts of the provided code as described in this handout and in the code itself.

10.1 Local Feature Extraction (30%)

a) Feature detection - feature points on a grid (10%)
Implement a very simple local feature point detector: points on a grid. Within the function `grid_points`, compute a regular grid that fits the given input image, leaving a border of 8 pixels in each image dimension. The parameters `nPointsX` $= 10$ and `nPointsY` $= 10$ define the granularity of the grid in the x and y dimensions. The function shall return `nPointsX` $\cdot$ `nPointsY` 2D grid points. The function to be written takes the following form:

```
vPoints = grid_points(img,nPointsX,nPointsY,border)
```

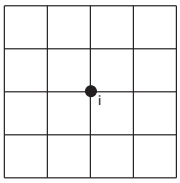

Figure 1: Grid point *i* and surrounding cells.

b) Feature description - histogram of oriented gradients (20%)

Next, we describe each feature (grid point) by a local descriptor. We will use the well known histogram of oriented gradients (HOG) descriptor. Implement the function `descriptors_hog` which should compute one 128-dimensional descriptor for each of the N 2-dimensional grid points (contained in the $N \times 2$ input argument `vPoints`). The descriptor is defined over a 4×4 set of cells placed around the grid point *i* (see Fig. 1).

For each cell (containing `cellWidth` $\times$ `cellHeight` pixels, which we will choose to be `cellWidth` $=$ `cellHeight` $= 4$), create an 8-bin histogram over the orientation of the gradient at each pixel within the cell. Then concatenate the 16 resulting histograms (one for each cell) to produce a 128 dimensional vector for every grid point. Finally, the function should return a $N \times 128$ dimensional feature matrix and also a $N \times (16 * 16)$ matrix, containing the image patch spanned by these cells. The latter matrix stores one image patch 16×16 per row and is needed later for visualization. The function to be written takes the following form:

```
[descriptors,patches] =
descriptors_hog(img,vPoints,cellWidth,cellHeight)
```

To construct the codebook, do the following:

10.2 Codebook construction (15%)

In order to capture the appearance variability within an object category, we first need to construct a visual vocabulary (or *appearance codebook*). For this we cluster the large set of all local descriptors from the training images into a small number of visual words (which form the entries of the codebook). For this, you will use the popular K-means clustering algorithm. This algorithm performs the following basic steps:

- Initialize each of the `k` cluster centers with the position of a randomly chosen feature from the set of input features `vFeatures`.
- Repeat for `numiter` iterations.
  - Assign each input point to the closest cluster center.
  - Shift the position of each cluster center to the mean of its assigned points.
- Return the `k` cluster centers.

The K-Means clustering algorithm is already efficiently implemented in Matlab, even with a better initialization strategy called K-means++. The function can be used in the following form:

```
[~,vCenters] = kmeans(vFeatures,k)
```

a) Create codebook (15%)

Fill the function `create_codebook` which takes as input the name of directories, loads each image in it in turn, computes grid points for it, extracts descriptors around each grid point, and clusters all the descriptors with K-Means. It then returns the found cluster centers. Make sure to use all descriptors from all images in both directories (positive and negative samples). The function to be written takes the following form:

```
vCenters = create_codebook(nameDirPos,nameDirNeg,k,numiter)
```

b) Visualize codebook

You can visualize the elements (visual words) of the codebook using the function `visualize_codebook` which is provided in the release. This function is invoked by `create_codebook` after the codebook is built. Each visual word is in fact a cluster center returned by the function `kmeans`. Ideally some visual words would represent a part of a car and some common features from images where a car is not present, e.g. a road.

10.3 Bag-of-words image representation (15%)

Using the appearance codebook created at the previous step we will represent each image as a histogram of visual words. This representation, also known as the **bag-of-words** representation, records which visual words are present in the image.

a) Bag-of-Words histogram (10%)

Fill the function `bow_histogram` that computes a bag-of-words histogram corresponding to a given image. The function should take as input a set of descriptors extracted from one image and the codebook of cluster centers. To compute the histogram to be returned, assign the descriptors to the cluster centers and count how many descriptors are assigned to each cluster (i.e. to each 'visual word'). The function to be written takes the following form:

```
bowHist = bow_histogram(vFeatures, vCenters)
```

b) Processing a directory with training examples (5%)

Fill the function `create_bow_histograms` that reads in all training examples (images) from a given directory and computes a bag-of-words histogram for each. The output should be a matrix having the number of rows equal to the number of training examples and number of columns equal to the size of the codebook (number of cluster centers). The function to be written takes the following form:

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

Using this function we process all positive training examples from the directory **cars-training-pos** and all negative training examples from the directory **cars-training-neg**. We collect the resulting histograms in the rows of the matrices `vBoWPos` and `vBoWNeg`.

10.4 Nearest Neighbor Classification (10%)

We first build a bag-of-words image classifier using the nearest neighbor principle. For classifying a new test image, we compute its bag-of-words histogram, and assign it to the category of its nearest-neighbor training histogram. Hint: use `knnsearch` function in Matlab.

a) Classifier (10%)

Fill the function `bow_recognition_nearest` that compares the bag-of-words histogram of a test image to the positive and negative training examples and returns the label 1 (car) or 0 (no car), based on the label of the nearest-neighbour. The function to be written takes the following form:

```
label = bow_recognition_nearest(histogram,vBoWPos,vBoWNeg)
```

10.5 Bayesian Classification (30%)

In the previous section, a simple nearest neighbor classifier was used to determine whether a given image contained a car or not. The histogram was used to describe the contents of the image. In this section, we develop a probabilistic classification scheme based on Bayes' theorem, as an alternative to the nearest neighbor classifier.

Let the probability that a car is present given a histogram be: $P(Car|hist)$. The likelihood of generating a histogram given that a car is observed: $P(hist|Car)$. The prior probability of observing a car in any image is $P(Car)$ and that of observing a histogram in any image is $P(hist)$. Bayes' theorem then states the following:

$$P(Car|hist) = \frac{P(hist|Car) * P(Car)}{P(hist)} \tag{1}$$

The denominator can be rewritten as follows (where !Car means that no car is present):

$$P(hist) = P(hist|Car) * P(Car) + P(hist|!Car) * P(!Car) \tag{2}$$

Replacing this in Eqn. 1 gives the posterior probability of a car given a histogram:

$$P(Car|hist) = \frac{P(hist|Car) * P(Car)}{P(hist|Car) * P(Car) + P(hist|!Car) * P(!Car)} \tag{3}$$

Conversely,

$$P(!Car|hist) = \frac{P(hist|!Car) * P(!Car)}{P(hist|Car) * P(Car) + P(hist|!Car) * P(!Car)} \tag{4}$$

For the classification task at hand, we need to evaluate $P(Car|hist)$. To compute this posterior probability, we first need to estimate the likelihoods $P(hist|Car)$, $P(hist|!Car)$ and prior $P(Car)$. This is done in the training stage described below.

a) Training (20%)

Recall that the observed histograms from the training images are stored as rows in the `vBoWPos` and `vBoWNeg` matrices. A row in `vBoWPos` corresponds to the `k` dimensional histogram of a particular positive training image j. For image j, each column i in its histogram corresponds to the number of times visual word i is observed in image j and we denote it by `vBoWPos`(j, i).

i. Let $X(i)$ be the distribution of the values of column i of `vBoWPos` (i.e. the distribution of the counts for visual word i over the positive training set). $X(i)$ is assumed to follow a normal distribution $N(\mu_p(i), \sigma_p(i))$, where $\mu_p(i)$ is the mean and $\sigma_p(i)$ is the standard deviation of $X(i)$. These can be estimated using the values in column i of `vBoWPos`. Fill in the function `computeMeanStd` which computes the mean and standard deviation for each column of the matrix `vBoW`. The function to be written takes the following form:

```
[mu sigma] = computeMeanStd(vBoW)
```

Use the function `computeMeanStd` to compute $\mu_p(i)$ and $\sigma_p(i)$ from the matrix `vBoWPos`.

ii. We now consider the case of `vBoWNeg`. Let $Y(i)$ be the distribution of the values of column i of `vBoWNeg`. $Y(i)$ follows $N(\mu_n(i), \sigma_n(i))$, where $\mu_n(i)$ is the mean and $\sigma_n(i)$ is the standard deviation of $Y(i)$. Use the function `computeMeanStd` to compute $\mu_n(i)$ and $\sigma_n(i)$ from the matrix `vBoWNeg`.

b) Testing (10%)

i. Let $U(i)$ be the entry in column i of the histogram of a test image that has to be classified. The probability of observing $U(i)$ in a car image (according to the positive training images) is denoted by $P(U(i)|N(\mu_p(i), \sigma_p(i)))$. This is a partial likelihood (as it's only for visual word i).

ii. Similarly, $P(U(i)|N(\mu_n(i), \sigma_n(i)))$ denotes the probability of observing $U(i)$ in the set of negative training images.

iii. Compute the joint likelihoods $P(hist|Car)$ and $P(hist|!Car)$ using the partial values computed in the previous two steps and assuming that the words appear independently as:

$$P(hist|Car) = \prod_{i=1}^{K} P(U(i)|N(\mu_p(i), \sigma_p(i))) \tag{5}$$

$$P(hist|!Car) = \prod_{i=1}^{K} P(U(i)|N(\mu_n(i), \sigma_n(i))) \tag{6}$$

iv. Assuming for simplicity $P(Car) = P(!Car) = 0.5$, evaluate the posterior probability $P(Car|hist)$ as in 3.

v. Make a decision on the presence/absence of a car using the following decision rule: if $P(Car|hist) > 0.5$, a car is present in the image.

Fill in the function `bow_recognition_bayes` that contains all the testing stages enumerated before and outputs a binary value: 0 - in the absence of a car, 1 - in the presence of a car. The function to be written takes the following form:

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

10.6 Hand in

Write a short report describing the pipeline and your implementation. Briefly answer the following questions:

- What is your classification performance using the nearest neighbour classifier?
- What is your classification performance using the Bayesian classifier? Is it better or worse? How do you explain this difference?
- Vary the number of cluster centres in your codebook, `k`. How does your categorization performance vary with `k`? Can you explain this behaviour?

Compress your report and source code (but not the data!) into a zip file and submit to moodle.

10.7 Use your own dataset (bonus: up to 10%)

Acquire or assemble your own dataset of positive and negative training examples of images of cars or some other object, and test your classification performance. What is your performance with your dataset? Discuss possible reasons for any differences you observe.

Hints:

a) Exploit the following MATLAB functions: `mean`, `std`, and `normpdf`.

b) If a column i of `vBoWPos` or `vBoWNeg` contains only zeroes, the mean and the variance will be zero. To circumvent $P(U(i)|N(\mu(i),\sigma(i)))$ being undefined (i.e. Matlab will evaluate $N(\mu(i),\sigma(i))$ to `NaN`), skip the column i by assigning 1 to it in the product expression of Eqn. 5 and Eqn. 6.

c) Products of many probabilities may have very small values, causing numerical problems. Instead of directly evaluating a product, evaluate a sum of the logarithm of each probability term instead.

d) Following from the previous hint, in the denominator of Eqn. 2, you have to take the logarithm of a sum. However, for the classification decision, you may neglect the denominator altogether as it appears in both $P(Car|hist)$ and $P(!Car|hist)$. Then the decision rule simply becomes $P(hist|Car) * P(Car) > P(hist|!Car) * P(!Car)$

e) Be aware of random and deterministic parts of the code. The initialization of the K-means clustering algorithm is random. Hence when evaluating the performance of the classifier with varying `k`, it is important to repeat the test multiple times for each `k` with a different random number seed each time.