Computer Vision Assignment 10: Object Category Recognition

Charlotte Moraldo

1. Local Feature Extraction

1.1 Feature detection: feature points on a grid

The local feature point detection is done with a very simple method: points on a grid. To implement it, I filled in the function `vPoints = grid_points(img,nPointsX,nPointsY,border)`,where `nPointsX` and `nPointsY` are set to 10 and define the granularity of the grid in the x and y dimensions, `border` gives the number of pixels we leave on each side of the image, and `vPoints` contains the coordinates of all the points on the grid.
To define the points of the image that will correspond to the grid points, I get rid of the border, and separate the remaining area given the required resolution. I then simply store the grid points coordinates in a `nPointsX*nPointsY*2` matrix.

1.2 Feature description: histogram of oriented gradients

Each feature (corresponding to grid point) has now to be described by a local descriptor, which is done using histogram of oriented gradients (HOG). This is implemented in the function `[descriptors,patches] = descriptors_hog(img,vPoints,cellWidth,cellHeight)`, where `vPoints` is computed in the section above, and `cellWidth`, `cellHeight` are set to 4.
Before computing the descriptors and patches, some preliminary steps are necessary. Firstly, the gradients in x and x of the image are calculated by doing `[grad_x,grad_y]=gradient(img)`. Then, their orientation can be found by doing: `theta = atan2(grad_y,grad_x)`. Now that we have the oriented gradients, we can compute their histogram.
The HOG algorithm is implemented in 3 main steps. For each grid point:

- I create a set of cells of dimension `cellWidth x cellHeight` (4x4) around the current grid point. This set of cells corresponds to a local patch, stored in the output variable `patches`.
- For each cell (containing `cellWidth x cellHeight` pixels), I compute an 8-bin histogram. This is done using the Matlab function `histcounts`, to which I input the orientation gradient of the current cell (computed in the preliminary steps). I also specify the edges of the histogram, 8 points equally spaced between $-\pi$ and π.
- Finally, I concatenate the 16x8 histograms into a 128-dimensional vector, which I return in the variable `descriptors`.

2. Codebook Construction

An appearance codebook is like a visual vocabulary, which we will use to capture the appearance visibility within an object category. To construct this codebook, I implemented the following:

- **Nearest-Neighbor Matching**: Given two sets of feature descriptor vectors D_1 and D_2, the function `findnn` computes for each feature vector in D_1 the nearest neighbor in D_2 (euclidian distance). In order to limit the use of `for` loops, I used the Matlab command `repmat`.

- **K-Means Clustering:** Given the input set of features `vFeatures`, the number of iterations `numiter`, and the number of desired clusters `k`, I can perform the k-means clustering algorithm. I firstly begin by initializing the center of each cluster to a different random feature from `vFeatures`, using the command `randperm`. Then, in a loop that is executed `numiter` times, I assign each point to the closest cluser, using the function `findnn`, detailed in the point above. The last step of the algorithm is to shift each cluster's center to the mean of its assigned points. Finally, we return in the variable `vCenters` the coordinates of the `k` cluster's center.

Once this is done, I completed the function `create_codebook` by calling in the correct order and with the correct input all the functions previously defined. Firstly, for each image, I collect the local feature points and compute a descriptor for each local feature point (`grid_points`), and then I create the HOG descriptors and patches (`descriptors_hog`). Once this is done for each image, I can call the function that will perform the k-means algorithm.
The provided code allows to visualize the codebook created, as shown in the figure below:

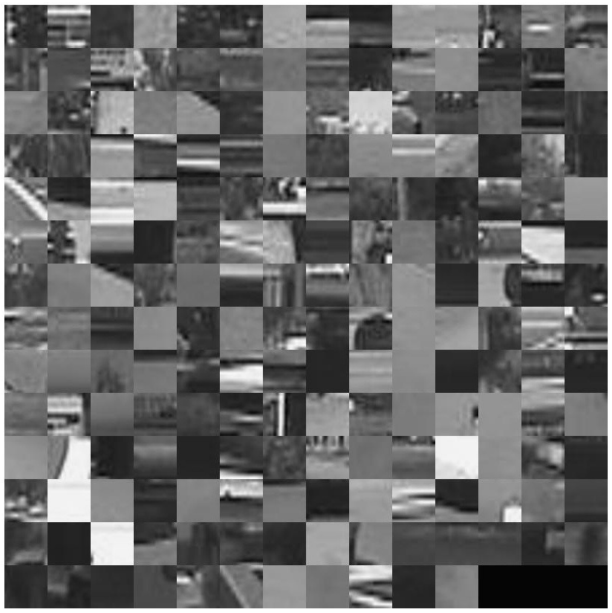

Figure 1: Codebook with k=200

3. Bag-of-words Image Representation

Now that we have the appearance codebook, we can represent each image as a histogram of visual words, known as bag-of-words representation. The first step is to compute the bag-of-words histogram corresponding to a given image. This is done in the function `bow_histogram`, which takes as input the previously computed variables `vFeatures`, the set of descriptors extracted from one image, and `vCenters`, the cluster centers.
I then match all the features to the codebook of cluster centers by looking at the smallest Euclidian distance between feature and center. Then, I count the number of matches for each codebook entry and save it in the activation histogram variable `histo`.

Finally, to create the BoW activation histogram for all the training examples from a given directory, I complete the function `create_bow_histograms`. Similarly as when creating the codebook, for all the images extracted from the directory, I firstly collect the local feature points and compute a descriptor for each local feature point (`grid_points`), then I create the HOG descriptors and patches (`descriptors_hog`). I can then fill in the vector `vBow`. Each line of this variable contains the output of the function `bow_histogram`, which corresponds to the BoW activation histogram, for a given image.

4. Nearest-Neighbor Classification

During training, we represented each image as a histogram of visual words. But during testing, in order to classify a new test image, we compute its bag-of-words histogram, and assign it to the category of its nearest-neighbor training histogram. The BoW histogram of the test image is compared to the positive and negative training examples, using the function `findnn`. If the distance to the positive examples is smaller than the one to the negative examples, then the label 1 is returned. This label indicates that a car has been recognized in the test image. On the opposite, if the distance to the negative example is smaller, then the label is set to 0: no car has been found in the test image.
The results and the accuracy obtained with the this classification method will be discussed in section 6.

5. Bayesian Classification

As an alternative to a simple nearest-neighbor classification, we implement a probabilistic classification scheme based on Bayes' theorem. The idea is to classify an image based on the posterior probabilities:

$$P(Car|hist) = \frac{P(hist|Car) \cdot P(Car)}{P(hist)} = \frac{P(hist|Car) \cdot P(Car)}{P(hist|Car) \cdot P(Car) + P(hist|!Car) \cdot P(!Car)}$$

$$P(!Car|hist) = \frac{P(hist|!Car) \cdot P(!Car)}{P(hist)} = \frac{P(hist|!Car) \cdot P(!Car)}{P(hist|Car) \cdot P(Car) + P(hist|!Car) \cdot P(!Car)}$$

Where $P(Car|hist)$ is the probability that a car is present given a histogram. $P(Car)$ is the prior probability of observing a car in any image and that of observing a histogram is $P(hist)$; $P(hist|Car)$ is the likelihood that the image contains a car given the observed histogram.

Firstly, we want to estimate the likelihoods $P(hist|Car)$, $P(hist|!Car)$ and the prior $P(Car)$.To do so, I begin by completing the function `computeMeanStd(vBoW)`. The distribution of the counts for visual word over the positive (negative) training set is given by the columns of `vBoWPos` (`vBoWNeg`), and is assumed to follow a normal distribution $N(\mu_{p,n};\sigma_{p,n})$. By doing `mean(vBoW)` and `std(vBoW)` in Matlab, I compute the mean and standard deviation for each column of `vBoW`.
Then, in the function `bow_recognition_bayes`, I calculate the probability of appearance of each word in the observed histogram, according to the normal distribution in each of the positive and negative bag of words. To do so, I loop over all the visual words to compute the log of their normal probability density function (command `normpdf`). After checking that what `normpdf` returns isn't NaN, I sum these probabilities to finally obtain $P(hist|Car)$ and $P(hist|!Car)$. In order to it more robust, I implemented the following: whenever the variance of `vBoWPos` or `vBoWNeg` is lower than 0.5, I set it back to 0.5. This way, it will circumvent the probabilities being undefined and will set the variance to a minimum default value.
Given that $P(Car) = P(!Car) = 0.5$, I can therefore finally compute the posterior probabilities $P(Car|hist)$ and $P(!Car|hist)$, defined above. The decision on the presence or absence of a car is made by simple thresholding: if $P(Car|hist) > P(!Car|hist)$, then we can set the label to 1, meaning that a car has been recognized in the image. On the opposite, if $P(Car|hist) < P(!Car|hist)$, then there is no car in the image.

6. Result Comparison

As pointed out in the hints, the initialization of the k-means algorithm is random. Looking at the accuracy of our two classifiers doesn't make much sense if the experiment is only run once. Therefore, I decided to run the experiment with 10 iterations in order to get a better representation of the classifiers' performance. Furthermore, it is also interesting to vary the number of clusters k in the codebook.
The plot below (fig.2) shows the results obtained when varying k from 25 to 300, while performing 10 tests for each classifier and for each value of k. The blue (red) 'x' markers show the accuracy obtained for Nearest-Neighbor (Bayes) classification, and the full lines show their mean.

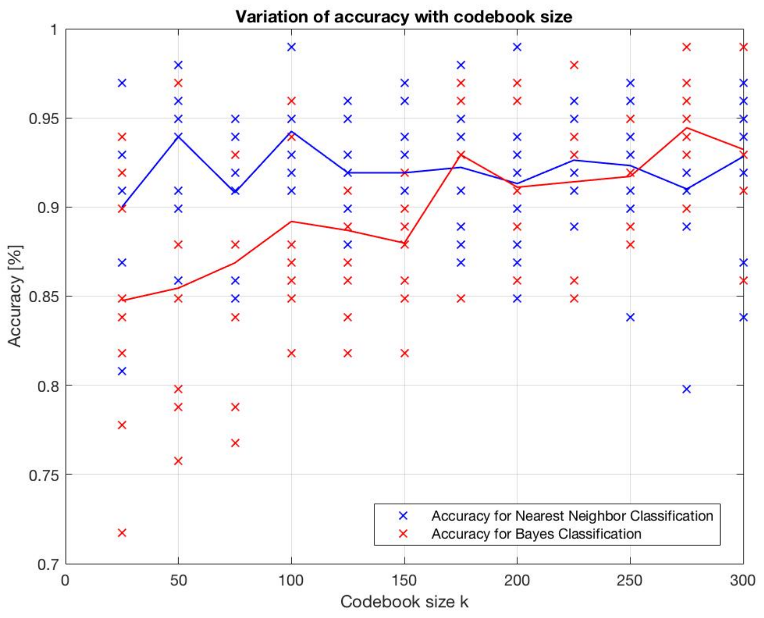

Figure 2: Categorization performance when varying the number of clusters k in the codebook

The mean obtained over all the experiments is the following: **92.93%** for **Nearest-Neighbor**, and **90.91%** for **Bayes**.

The NN method is slightly more accurate. Indeed, it has a much higher complexity and can therefore only be used with datasets that aren't too big. The Bayesian classifier is much faster but assumes conditional independence, resulting in diminished efficiency in this case. However, in more complicated cases like categorizing news, face detection, or email spam filtering, Bayes' classification is more appropriate.
When varying k, the accuracy of the NN classifier stays relatively constant, as what is seen on the graph. For the Bayesian classifier, the accuracy increases slightly with k. This could be explained by the fact that a small k won't have enough words to describe different parts of the car, and therefore an image with containing a car in a different angle won't be recognized. Starting k=200, the accuracy of both methods is almost the same.