Computer Vision

Alessia Paccagnella

December 11, 2019

1 Categorization

1.1 Local Feature Extraction

1.1.1 Feature detection - feature points on a grid

For this task, we had to implement the function `grid_points`, to compute a regular grid that fitted the given image. The function took as input `nPointsX` and `nPointsY`, representing the granularity into the x and y dimensions, the `border` and the `img`. The border was used to leave a certain amount of pixels above the each image dimension in the grid. I removed the border from both of the dimensions of the image and then divided the remaining area as required, creating equally distanced points in both of the directions.

1.1.2 Feature description - histogram of oriented gradients

Every feature, or grid point, has to be described by a local descriptor.We are asked to implement the HOG descriptor. For every N grid point contained in the input `vPoints` I computed a 128 dimensional descriptor. It is defined over `cellWidthxcellHeight` set of cells around every grid point. In our case, `cellWidth` and `cellHeight` are equal to 4.
For for all the local feature points, I entered a loop in which I created a 4x4 cell (`patch`). For each cell I created an 8-bin histogram over the orientation of the gradient at each pixel within the cell. I used the matlab function `histcounts` to do that. In the given code, we already had the gradient vectors and the direction of it calculated.
I used the function `linspace` to equally spread 8 points between $-\pi$ and π and use them as edges for the histogram.
After this, I concatenated the histogram vector (which has dimensions 16x8) to be 1x128 for every grid point, and added it to the final vector to be returned.
The Nx(16x16) matrix that I also returned corresponds to the `patches` matrix, which contains all the local patches.

1.2 Codebook construction

1.2.1 Create codebook

To create a codebook, I computed the grid points by calling the function written in the previous task `grid_points`. Then I extracted the descriptors, always with the function implemented before

`descriptors_hog`. Finally I called the function `kmeans1` to cluster all the descriptors, obtaining the found cluster centers. Inside the function `kmeans1`, I call the usual `kmeans` function of Matlab. This function, for numiter iterations, finds the closest cluster for each of the points of `vFeatures` in `vCenters`. Then it shifts the center of the cluster to the mean of the points of the cluster.

1.2.2 Visualize codebook

To visualize the elements of the codebook I called the provided function `visualize_codebook` after the codebook is built.

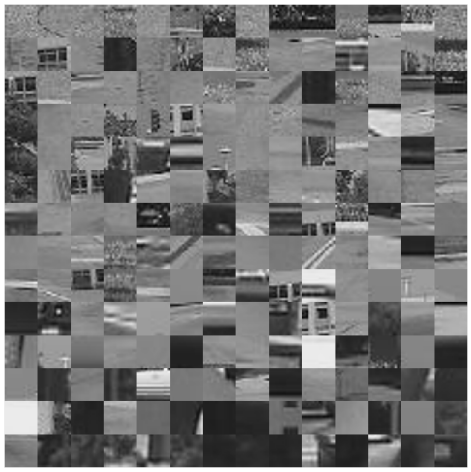

Figure 1: Visualization of the codebook with k = 200

1.3 Bag of words image representation

1.3.1 Bag of Words histogram

Each image can now be represented as a histogram of visual words. To compute a bag-of-words histogram, I filled the `bow_histogram` function. The function took as input a set of descriptors extracted from the image (`vFeatures`)and the codebook of cluster centers (`vCenters`). I assigned the descriptors to the cluster centers by calculating the minor distance between the feature and the centers, and counted how many descriptors were assigned to each cluster.

1.3.2 Processing a directory with training examples

I filled the function `create_bow_histograms`. The function, with the provided code, reads in all training examples images from a given directory. It creates the grid, compute the descriptors and then calls the previous implemented function `bow_histogram` computing a bag-of-words histogram for each image.

1.4 Nearest Neighbor Classification

In order to classify the images with the label 1 (a car is present in the image) or 0 (there is no car in the image), I filled the function `bow_recognition_nearest` that for each vector in the

histogram calculates the nearest neighbor in the positive and negative training examples. If the distance with the positive example is minor than the one with the negative example, then there is a car in the image. Contrarily, there is no car.

1.5 Bayesian Classification

We now want to use a different classifier to determine whether a given image contains a car or not. Therefore we develop a probabilistic classification scheme based on Bayes' theorem.
We assume that the probability of presence of a car is, given a histogram,

$$P(Car|hist)$$

The probability of generating that histogram given the presence of a car is

$$P(hist|Car)$$

the probability of observing a car in an image is

$$P(Car)$$

and the probability of observing a histogram in an image is

$$P(hist)$$

Bayes' theorem says:

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

where

$$P(hist) = P(hist|Car)P(Car) + P(hist|!Car)P(!Car)$$

.

1.5.1 Training

For the training part, we have two matrixes containing the training images called `vBoWPos` and `vBoWNeg`. Every row of `vBoWPos` corresponds to the histogram of a specific positive training image j. Every column i for image j indicates the number of times a visual word i is observed in image j. By assuming that X(i) is the distribution of the values of column i of the matrix `vBowPos`, and that it follows a normal distribution, we want to estimate the mean and the standard deviation of it. I did it inside the function `computeMeanStd`, by using the Matlab functions `std` and `mean`. The same reasoning works when considering the case of `vBoWNeg`.

1.5.2 Testing

Finally, we want to fill the function `bow_recognition_ bayes` that outputs 0 if the car is not present, 1 if it is. The aim is calculating the probability of appearance of each word in the histogram. Therefore I iterate over all the visual words, compute the log of their pdf, which I calculate by using the Matlab function `normpdf`, and sum it to the probability. When out of the cycle, I did the exponential version of the two variables, `pCar` and `pNotCar`, in order to go back to the previous base.

2 Questions

2.1 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?

I did 5 trials for each k that I used, and took the mean of them for both of the techniques. I repeated it for more than once cause, as written in the assignment, the initialization of the k-means algorithm is random. These are the means calculated from results that I obtained.

Method	k = 300	k=200	k=100	k=50	k = 15
NN	0.862	0.909	0.949	0.961	0.949
B	0.967	0.976	0.963	0.934	0.925

Here the stds are shown.

Method	k = 300	k=200	k=100	k=50	k = 15
NN	0.020	0.025	0.031	0.013	0.019
B	0.023	0.012	0.004	0.012	0.021

What I first noticed is that with higher values of k (major than 100), I obtain a better result with the Bayes classification method then the nearest neighbors one.
However, by lowing the value of k, by using for example 50, we can see that the nearest neighbor method obtains a better mean accuracy than Bayes. As the complexity of the space grows (therefore with bigger ks), the accuracy of the k algorithm comes down: this is because it would need more data to be accurate, but by using more data the order of this classifier increases and it becomes too slow. The optimal k for which I obtained the best result with NN was k = 50.
The Bayes method is much faster than NN, especially with big datasets. It could be chosen for real time prediction. We see that the accuracy for the bayes classifier increases with k. Small k could result in not having enough visual words to describe parts of the car, therefore leading to minor accuracy.