Computer Vision Assignment 10

Luca Ebner (ebnerl@student.ethz.ch)

December 20, 2020

In this weeks assignment we program an algorithm that does Image Categorization. More specifically, we aim for an algorithm that recognizes images that contain a car (back view) and distinguish it from images where no car is present.

The algorithm is based on a Bag-of-Words histogram representation of images. The key idea behind it is to recognize features in a given image and assign them to *visual words* from a vocabulary list. Following, each image can be described by counting which visual words from the list occur how many times.

1. Local Feature Extraction

In order to describe an image with *visual words*, we first need a definition on how such a word looks like. In this exercise, we use the so called Histogram of Oriented Gradients (HOG) descriptor. The HOG descriptor for the image patch around a given pixel is defined over a 4×4 set of cells around the pixel, where in each cell the orientations of the image gradient is stored in histograms with bin size 8. This in turn results in a descriptor of size $4 \times 4 \times 8 = 128$ histogram, the *visual word*, for each feature point in an image. The complete image is then described by a $N \times 128$ matrix, where N is the number of feature points.

In this exercise, we extract feature points from a given image by simply taking the pixels from a pre-defined grid. For this, we implement a function

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

which takes the image and grid size parameters as inputs and returns a $N \times 2$ matrix for the resulting N 2D point coordinates. For these N points on the grid we then create a function

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

which takes the image, point coordinates and HOG cell dimensions as inputs and returns a $N \times 128$ matrix which contains all local feature descriptors for the image as well as the image patches corresponding to the features (used for visualization).

2. Codebook contruction

In order to teach our algorithm how to recognize car images, we need to create a visual vocabulary (the *appearance codebook*) to capture the variability of appearance for the car category. For this, we create a function to cluster the set of all visual words from the training set and come up with a the appearance codebook for the car category:

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

The function takes the directory paths to the positive and negative training sets as well as the number of visual words to find as inputs and returns the computed histogram clusters. For the clustering itself, the Matlab built-in function `kmeans` was used. The visualization of such a resulting appearance codebook can be seen in Figure 1.

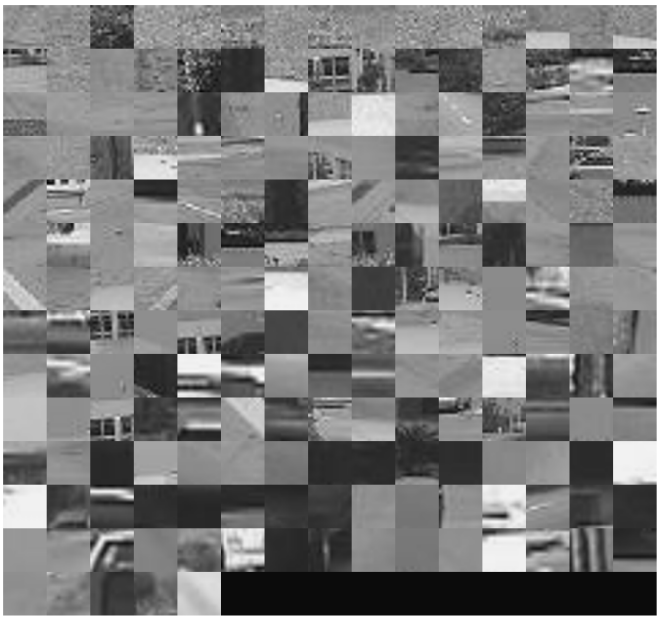

Figure 1: Visualization of an appearance codebook with 200 visual words.

3. Bag-of-words Image Representation

Once we have created our appearance codebook, we want to describe our test images with the vocabulary from the codebook. The so called *Bag-of-Word* (BoW) histogram of each image should contain a count on how many times each visual word from the codebook occurs in the image. For this, we implement another Matlab function:

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

The function takes as inputs the feature descriptors from the grid points of the image as well as the cluster centers (the visual word vocabulary). Since the feature descriptors from the image will of course not exactly match the visual words from the appearance notebook, they are assigned to the nearest cluster center found by the Matlab function `knnsearch`. As output the function computes a $1 \times k$ histogram, where k is the number of visual words in the codebook.

In order to train our algorithm, we create the function

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

which computes the BoW representation with the `bow_histogram` function for each of the images in a given directory, given the appearance codebook. We use this function to compute all the BoW histograms for the positive and negative train set and store these histograms as `vBoWPos` and `vBoWNeg`. We can later compare our test image BoW histogram to the histograms found in the test sets to determine if we are looking at "car" or "no car" image.

4. Nearest-Neighbor Classification

There are several ways how we can compare our image BoW histogram to our training set BoW histograms. One of the easiest ways to do that is to do a nearest neighbor comparison. We can guess the label of an image by finding the most similar histogram from the training sets. If the most similar histogram is one that stems from the "car" training set, it is likely that the image we are looking at does in fact also show a car and vice versa. With the function

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

we can give the image histogram and the "car"/"no car" training histograms as input and get the nearest neighbor guessed label as output. To compute the Nearest-Neighbor histogram, we can again use the Matlab function `knnsearch`.

5. Bayesian Classification

Apart from the deterministic Nearest-Neighbor classification, the Bayesian classification offers a more probabilistic approach that calculates the probabilities $P(car|histogram)$ and $P(!car|histogram)$. The label is then assigned depending on which probability is greater. In order to compute these probabilities, we can make use of Bayes' theorem:

$$P(car|histogram) = \frac{P(histogram|car) * P(car)}{P(histogram)} \tag{1}$$

$$P(!car|histogram) = \frac{P(histogram|!car) * P(!car)}{P(histogram)} \tag{2}$$

In this exercise it is given that $P(car) = P(!car) = 0.5$ and the denominator $P(histogram)$ is of course the same in both cases. To calculate the remaining probabilities, the visual words in the BoW histograms are assumed to follow a normal distribution with μ_i and σ_i equal to the mean and standard deviation of word i from the training set histogram lists `vBoWPos` and `vBoWNeg`. The probability $P(histogram|car)$ can be computed by the formulas

$$P(histogram|car) = \prod_{i=1}^{K} P(U(i)|N(\mu_{Pos}(i), \sigma_{Pos}(i))), \tag{3}$$

$$P(histogram|!car) = \prod_{i=1}^{K} P(U(i)|N(\mu_{Neg}(i), \sigma_{Neg}(i))), \tag{4}$$

where $U(i) \sim \mathcal{N}(\mu_{...}, \sigma_{...})$ is the probability of observing visual word i with its count in the set of positive or negative training images, respectively. To compute all this, we create the Matlab function

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

which again takes the image's BoW histogram and the "car"/"no car" training histograms as input and returns the guessed label.

6. Discussion

Figure 2 shows how the accuray of the Nearest-Neighbor and Bayesian approach compare side by side while varying the codebook size k and running the algorithm several times for each k.

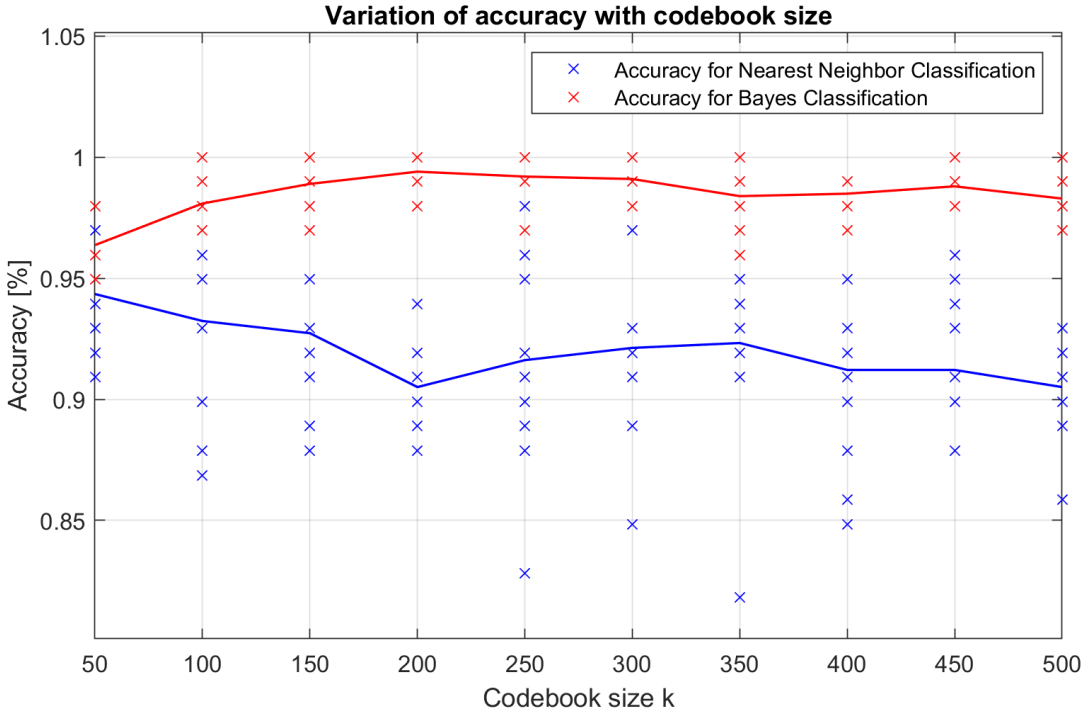

Figure 2: Side by side comparison between Nearest-Neighbor and Bayesian approach while varying $k = [50, 100, 150, 200, 250, 300, 350, 400, 450, 500]$ and collecting 10 data points each.

From the Figure above we can see that the mean accuracy for $k = 200$ (which is proposed by the exercise template) lies at about **0.91% accuracy for the Nearest-Neighbor approach** and **0.99% for the Bayesian classification**.
Furthermore, as we increase the codebook size, we can see that the accuracy slightly goes down in both cases, but more so in the Nearest-Neighbor approach. This could be explained by the following: If we increase our vocabulary size, the visual words tend to get more and more similar since the descriptor space still stays the same, which makes the algorithm more error prone. This theory can easily be checked by setting the codebook size to something like $k = 5000$. The resulting mean accuracy in this case lies at only 0.55 for Nearest-Neighbor and 0.66 for Bayes. In comparison to Nearest-Neighbor, the Bayes method does somewhat account for the similarity between visual words: It assumes the word count to follow a normal distribution, whereby the variance is calculated from the whole test sets. Since the counts for 'similar' words would vary a lot between the test images, the resulting normal distribution would be more forgiving.

In summary one could argue that for simple problems, the Nearest-Neighbor approach is more comfortable, because it is easy to implement and the performance is similar for small appearance codebooks. For more complex tasks with bigger codebooks however, the Bayesian classification might be more suited.