Computer Vision
Homework Assignment 10 : Image Categorization

Autumn 2018

Nicolas Marchal

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Vision
and Geometry Lab

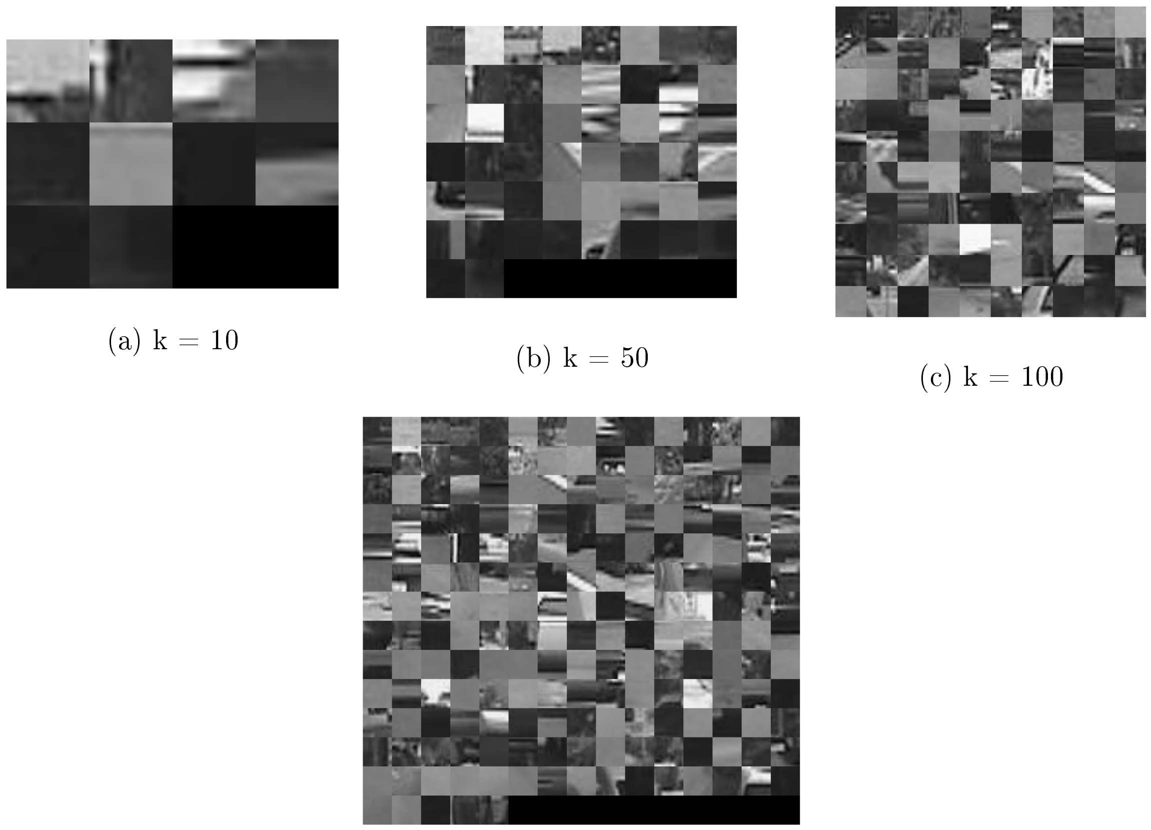
1 Local Feature Extraction

The function `grid_points` simply divides the immage as aa rectangular grid. Then, the function `descriptors_hog` uses the histogram of oriented gradients (HOG) to create a descriptor for each feature (= each grid point). This part is essential to the project, but I will not discuss it further as it is rather straightforward and well explained in the project description.

2 Codebook Construction

By extracting features in all the images of our training dataset, we end up with thousands of features. We want to reduce the feature space, using only really relevant features. These relevant features are known as our codebook. To select the most relevant features, we will do a nearest neighbours clustering. I can decide how many feature I will use in my codebook. The performance of the algorithm is linked to the number of features used for the codebook. This will be discussed later in more details.

The steps of the nearest neighbours algorithm are explained in the project description. I have completed the function `findnn`, `kmeans` and `create_codebook`. Finally the function `visualize_codebook` allows us to see the resulting features in our codebook. The results are shown below for different number of k means. Note that the fully black pixels on the bottom are not features, but they simply complete the figure such that it looks rectangular.


(a) k = 10

(b) k = 50

(c) k = 100

(d) k = 200

Figure 1: Codebooks for different k

3 Bag of Words

The function `bow histogram` simply assigns each feature of an image to the nearest one in the codebook (can reuse function `findnn`). With the function `create_bow_histograms` we will create the bag of words for all the images.

4 Nearest Neighbours Classifier

The nearest neighbour classifier is really intuitive. We take the feature vector (the number of features being the size of the codebook) and find the nearest neighbours in all the training examples. If this nearest neighbours is a car, we assign the label car, and vice versa.

The performance of this classifier will be evaluated later.

5 Bayesian Classifier

The baysian classifier is a probabilistic method. We need to calculate the mean and standard deviation of our features, which I did in the function `computeMeanStd`.

The final goal is to estimate the probability of a car, given a histogram: $P(car \mid hist)$. To do so we use bayes' theorem and we do some assumptions (such that the features are uncorrelated). In the function `bow_recognition_bayes` we then set a hard threshold and categorize an image as a car as long as $P(car \mid hist) > 0.5$.

We will analyse the performance of this method and suggest an amelioration in the next part of this report.

6 Performance increase and comparison of Results

Lets first study the nearest neighbours accuracy. By looking at figure 2a, we see that by implementing the method given in this project, we get a fairly high accuracy. The algorithm need a minimum amount of features (around 50) but the results seem to be more or less independent of the codebook size above this. The accuracy is around 95%.

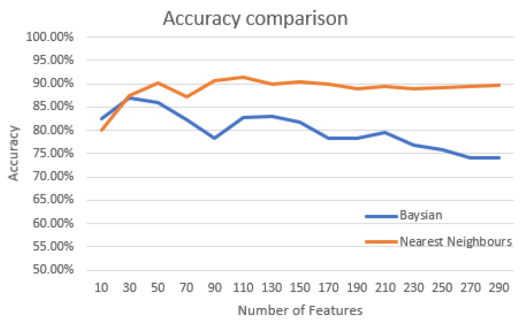
Now, lets study the Bayesian classifier. As this method is more complicated, and models probabilities rather than relying on one nearest neighbour, I expected the performance to be better than NN classifier. However, by looking at figure 2a we see that it is not the case.

I think that the issue comes from the fact that we have very few training images, and some of the features of the codebook have a very small variance. The case where the variance of a feature is zero has to be handled with caution. I decided to ignore the features with a variance of zero as it was the simplest approach (with a larger data set, we should not get zero variance on some features). This gives an accuracy that reaches 85% for 30 features, but it then decreses linearly down to 75 % with a codebook size of 290.

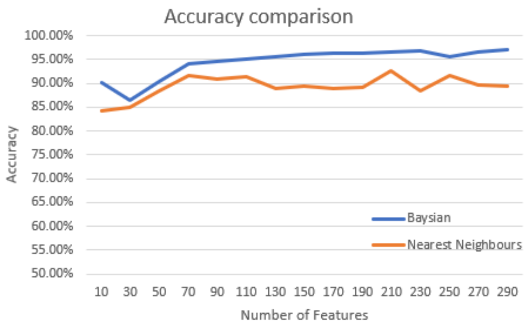
As the results were not satisfactory with my first approach, I tried a new one. I was persuaded that the results should be better than nearest neighbours, plus I thought that more features should increase the performance of the baysian classifier. It makes sense that the accuracy increases with the number of features, as when we have a lot of features it does not really affect the probability distribution is some of the features are not in the image. In practice all the features from a car can not be seen in all the car images (depending on the viewpoint etc) so increasing the number of feature is a good idea (although it should not be arbitrary high).

My very simple, yet efficient solution was to set the variance to 0.5 for all features who had a variance less than that. The idea is that when the variance is too small, it penalises a lot an image which does not have this exact feature and I want to avoid that. My choice of 0.5 was motivated by the fact that it gave nice results, but this arbitrary choice could be optimized further. Nevertheless, I now reach up to 98 % <u>on average</u> (for 290 dimension feature space). This is much better and we see that the accuracy increases with the number of features as expected.

Finally, I would like to point out that both methods have some probabilistic steps and the results can vary quite a lot between different tries. Therefore, the plots on figure 2 show the average accuracy over 15 experiments, for codebook size from 10 to 290 (with increment of 20).



(a) results with the method suggested in the exercise



(b) results with my new method

Figure 2: Results comparison