9 Computer Vision – Pierre Beckmann and Mathis Lamarre

9.1 Local Feature Extraction

The local feature point detector is a very straight-forward grid. To compute it, we calculate the space between two points in the x- and y-direction given the size of the image, the border we leave on each side and the granularity. Then we store the points' coordinates in a nPointsX*nPointsY x 2 matrix.

For the histogram of gradients (HOG) we added pi to all negative angles because it still represents the same line. This way our histogram is for values between zero and π. Then for every gradient we added to the two nearest angles in the histogram proportionally to the distance and for maximal precision. We also weigh the angles according to the magnitudes of corresponding gradients. The resulting histogram can now be used as a descriptor for cells of pixels.

9.2 Codebook Construction

We create a codebook using K-means clustering. First, we find the nearest neighbor (in Euclidean distance) between all the features and the set of k randomly assigned cluster centers using `repmat` to limit the use of for loops. Then, we shift all the cluster centers to the mean of the features that were assigned to them. We repeat this algorithm for a defined number of steps.

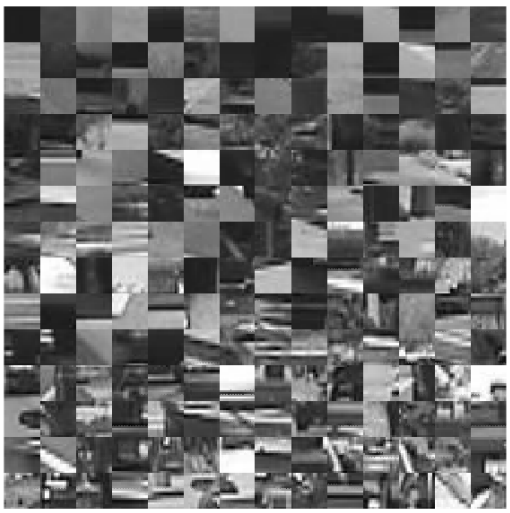

Figure 1: **Example of an obtained codebook.** This figure was obtained using given function `visualizecodebook.m`.

With all the car images provided, the output is the visual codebook seen in Fig.1. We do recognize some parts of a car, mainly the edges.

9.3 Bag-of-words Image Representation

In order to create a BoW histogram, for each match the features of the images with the codebook of cluster centers (smallest feature to center Euclidean distance) and count the matches for each cluster. Then, for all the images, we find the features, compute their HOG descriptors and create their BoW histogram.

9.4 Nearest Neighbor Classification

The nearest neighbor classifier finds the smallest distance between an image's BoW histogram and the histograms of the positive and negative sets. It assigns the image to the closest set.

9.5 Bayesian Classification

Next we implemented a probabilistic classification scheme based on Bayes' theorem.

First we implement a simple function `computeMeanStd` that returns μ and σ for each column of a matrix.

We then use this function to get the mean and standard deviation of the positive and negative observed histograms from the training images. We can now compute the joint likelihoods $P(hist|car)$ and $P(hist|!car)$ assuming that the words appear independently. We used the matlab function `normpdf` to do this and made sure to work with logarithms for numerical stability. Therefore we summed up the logs and took the exponential of the result instead of using a multiplication.

9.6 Results

To evaluate the performance of the two classifiers we took the mean of multiple tests because of the random and deterministic parts of the code.

With $k = 200$ clusters and 5 tests the nearest neighbor classifier yielded a mean of $95,56\%$ with a standard deviation of 0.049%. The bayesian classifier yielded a mean of $91,72\%$ with a standard deviation of 0.1389%.

The nearest neighbor method is slightly more accurate than the bayesian method. The nearest neighbor classifier has a much higher complexity and can therefore only be used with small enough data sets. The bayesian classifier is much faster but assumes conditional independence which makes it slightly less efficient in this case but much more appropriate for other cases such as email spam filtering.

With a very large k, the result are a bit less accurate maybe because there are so many words in the BoW that some of them are not relevant and don't represent the car. On the contrary, a small k won't have enough words to describe different parts of the car and an image with a slightly different view of a car won't be recognized. The optimal amount of clusters with this data set seems to be around 50 after some testing.