Computer Vision

Assignment 10: Image Categorization

Student: Ivan Alberico

1. Local Feature Extraction

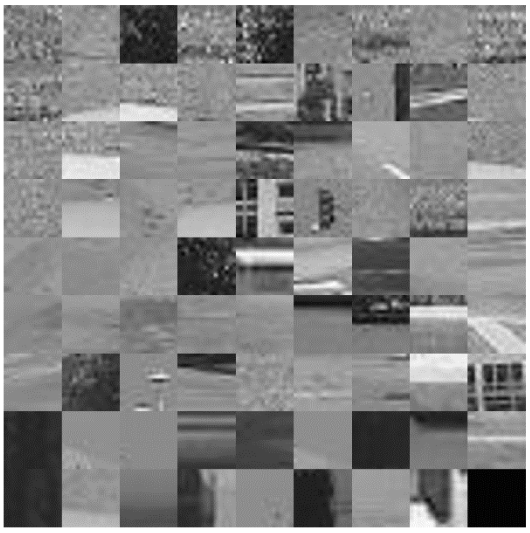
The first step in order to implement a Bag-of-Words image classifier is the extraction of local features. First, I implement a local feature point detector in the function `grid_points.m`, which computes a grid over an input image, according to a certain granularity along the x and y dimensions, and by leaving a specified border (in our case the border was set to 8 pixels). To do that, I first computed two vectors expressing the granularity along the x and y directions through the function `linspace` (given the parameters `nPointsX` and `nPointsY`) and then I used the function `meshgrid` in order to obtain the coordinates of all the points in the grid.

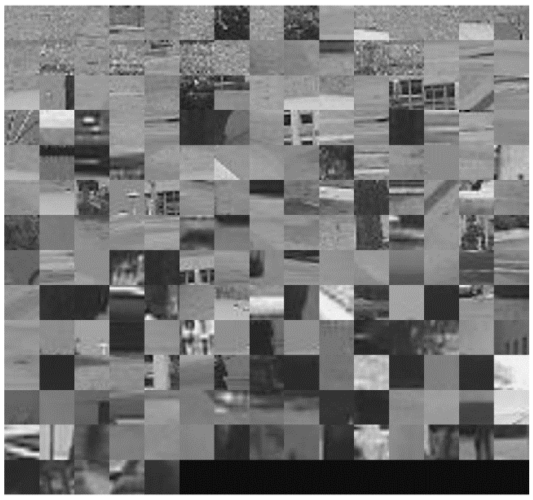
With this being done, I then needed to derive the local descriptors for each feature, namely for each point of the grid. HOG descriptors were used in this assignment, with each descriptor being a 128-dimensional array. For each point of the grid, a 4x4 set of cells was considered, each of them containing 4x4 pixels. For each cell, I computed the 8-bin histogram over the orientation of the gradient for the pixels in the specific cell (the orientation is basically the angle, computed with the atan2 function, between the gradient along the x and y directions). Besides computing the histogram for each cell, the function `descriptors_hog.m` also returns the image patch of the cell, which is used for the visualization of the local features descriptors.

2. Codebook construction

In order to construct the Codebook, I first extract the descriptors of all the images in our training dataset (both positive and negative images). After that, I apply K-means algorithm to these descriptors (which are stored in the vector `vFeatures`), according to a specified value of K. The K centers of these clusters are then shown through the function `visualize_codebook.m`, which makes use of the patches previously extracted from the image.

K = 36


K = 80



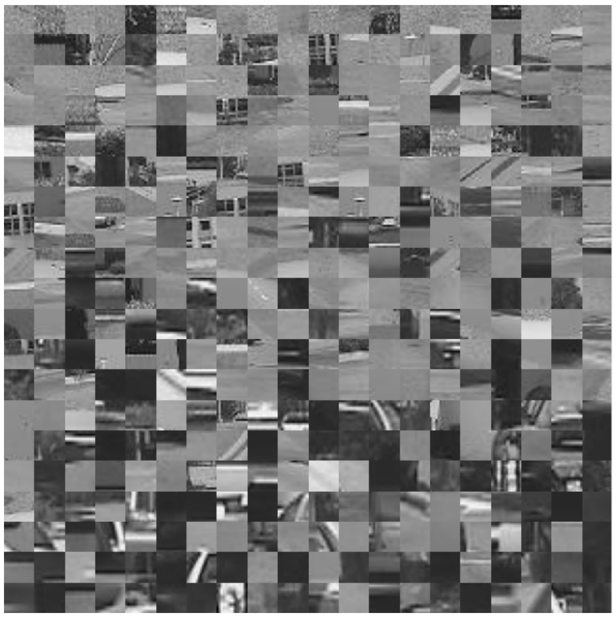


Figure 1 — The pictures represent the generated Codebooks for different values of K, namely K=36, K=80, K=200 and K=400.

3. Bag-of-words image representation

The Bag-of-Words algorithm treats each image of the dataset as a histogram of visual words. First of all, in the `bow_histogram.m` function, I compute the bag-of-words histogram of an image, given the cluster centers and the feature descriptors. To do that, for each feature descriptor, I compute the distance from all the cluster centers and I assign it to the cluster with respect to which it shows the smallest distance. The histogram is then computed by counting how many times we have assigned a feature to a specific cluster.

In the function `create_bow_histograms.m`, this procedure is repeated for any image in the training dataset, for both the positive and negative cases.

3.1 Nearest Neighbor Classification

Once the features of the training dataset have been assigned to the clusters, we need to train a classifier in order to make future prediction on the testing dataset. In this assignment, two types of classifications were implemented: Nearest-Neighbor classification and Bayesian classification.

As far as the first one is concerned, it is implemented in `bow_recognition_nearest.m` through the function `knnsearch`. We first compute the histogram of the test image and then we compare it to all the previously obtained histograms of the training dataset. We assign the image to label 1 and 0, according to which class belongs the smallest distance.

3.2 Bayesian Classification

The Bayesian classification, instead, classifies images according to probabilistic considerations on our data. Following the Bayes' theorem

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

the assignment of the each testing image to a class is done by looking at the posterior probability of observing Car or $!\,Car$, given the specific histogram.

If $P(Car|hist) \propto P(hist|Car)\,P(Car) > P(!\,Car|hist) \propto P(hist|!\,Car)\,P(!\,Car)$, then we assign the image to class 1 and vice versa. The prior distribution is assumed to be equal in both cases, namely $P(Car) = P(!\,Car) = 0.5$, while the likelihood distributions $P(hist|Car)$ and $P(hist|!\,Car)$ are assumed to be drawn from a normal distribution, with words being independent among each other.

4 Results

I will now present the results obtained for the two different cases. Since the initialization of the K-means algorithm is random, for each specific case I compared the results obtained by averaging over 10 iterations. In this way, more reliable conclusions could be drawn. Moreover, what was observed was that the performance of the algorithm strongly depended on the number of cluster K we set. We will now see how the overall performance changed with different values of K.

K = 25	NN classification accuracy (%)	Bayesian classification accuracy (%)
Iter 1	0.9495	0.8889
Iter 2	0.8889	0.9697
Iter 3	0.9394	0.9091
Iter 4	0.9091	0.9192
Iter 5	0.9293	0.9192
Iter 6	0.9091	0.8889
Iter 7	0.9596	0.9192
Iter 8	0.8788	0.9091
Iter 9	0.9293	0.8788
Iter 10	0.9192	0.9091
Average	**0.9212**	**0.9111**

K = 50	NN classification accuracy (%)	Bayesian classification accuracy (%)
Iter 1	0.8990	0.9293
Iter 2	0.9495	0.8889
Iter 3	0.8788	0.8889
Iter 4	0.9394	0.8788
Iter 5	0.9293	0.9394
Iter 6	0.9293	0.8889
Iter 7	0.9192	0.8889
Iter 8	0.9899	0.8687
Iter 9	0.9697	0.9495
Iter 10	0.9091	0.9293
Average	**0.9313**	**0.9051**

K = 100	NN classification accuracy (%)	Bayesian classification accuracy (%)
Iter 1	0.9495	0.9394
Iter 2	0.9596	0.9293
Iter 3	0.9091	0.9192
Iter 4	0.9697	0.8687
Iter 5	0.9394	0.9697
Iter 6	0.9192	0.9596
Iter 7	0.9192	0.8384
Iter 8	0.9394	0.9293
Iter 9	0.8990	0.8586
Iter 10	0.9394	0.9596
Average	**0.9343**	**0.9172**

K = 150	NN classification accuracy (%)	Bayesian classification accuracy (%)
Iter 1	0.8687	0.9091
Iter 2	0.9394	0.9596
Iter 3	0.9495	0.9091
Iter 4	0.9495	0.9293
Iter 5	0.9091	0.9091
Iter 6	0.9394	0.9192
Iter 7	0.8182	0.9798
Iter 8	0.8788	0.8990
Iter 9	0.9495	0.9394
Iter 10	0.9091	0.8990
Average	**0.9111**	**0.9253**

K = 200	NN classification accuracy (%)	Bayesian classification accuracy (%)
Iter 1	0.9192	0.8687
Iter 2	0.8384	0.9091
Iter 3	0.8990	0.9091
Iter 4	0.9192	0.9596
Iter 5	0.8687	0.9394
Iter 6	0.9091	0.9798
Iter 7	0.8990	0.8990
Iter 8	0.9192	0.9495
Iter 9	0.9091	0.8788
Iter 10	0.9192	0.9293
Average	**0.8990**	**0.9212**

K = 250	NN classification accuracy (%)	Bayesian classification accuracy (%)
Iter 1	0.9091	0.8586
Iter 2	0.9192	0.8990
Iter 3	0.8990	0.8990
Iter 4	0.8889	0.9293
Iter 5	0.8788	0.8788
Iter 6	0.9394	0.9495
Iter 7	0.8889	0.8788
Iter 8	0.8990	0.8788
Iter 9	0.8788	0.9596
Iter 10	0.9495	0.9293
Average	**0.9051**	**0.9061**

K = 300	NN classification accuracy (%)	Bayesian classification accuracy (%)
Iter 1	0.8081	0.8081
Iter 2	0.9192	0.8384
Iter 3	0.8788	0.8889
Iter 4	0.8889	0.8889
Iter 5	0.9293	0.9091
Iter 6	0.8182	0.8990
Iter 7	0.8485	0.8889
Iter 8	0.9293	0.8889
Iter 9	0.8384	0.8384
Iter 10	0.9394	0.8889
Average	**0.8798**	**0.8737**

K = 500	NN classification accuracy (%)	Bayesian classification accuracy (%)
Iter 1	0.8687	0.8788
Iter 2	0.8889	0.8384
Iter 3	0.7980	0.7677
Iter 4	0.8485	0.7576
Iter 5	0.9091	0.7071
Iter 6	0.8485	0.7778
Iter 7	0.8990	0.7980
Iter 8	0.9091	0.8384
Iter 9	0.8586	0.7677
Iter 10	0.9192	0.6970
Average	**0.8747**	**0.7828**

K = 750	NN classification accuracy (%)	Bayesian classification accuracy (%)
Iter 1	0.8990	0.6364
Iter 2	0.8990	0.6465
Iter 3	0.8586	0.6768
Iter 4	0.7778	0.7273
Iter 5	0.8283	0.7879
Iter 6	0.9192	0.6465
Iter 7	0.8081	0.6364
Iter 8	0.7475	0.6667
Iter 9	0.8687	0.6061
Iter 10	0.8485	0.7374
Average	**0.8455**	**0.6768**

K = 1000	NN classification accuracy (%)	Bayesian classification accuracy (%)
Iter 1	0.8081	0.7475
Iter 2	0.8990	0.6061
Iter 3	0.8283	0.5455
Iter 4	0.9091	0.5657
Iter 5	0.8384	0.5960
Iter 6	0.8384	0.5455
Iter 7	0.9394	0.5758
Iter 8	0.8687	0.6162
Iter 9	0.7980	0.5455
Iter 10	0.9192	0.5859
Average	**0.8646**	**0.5929**

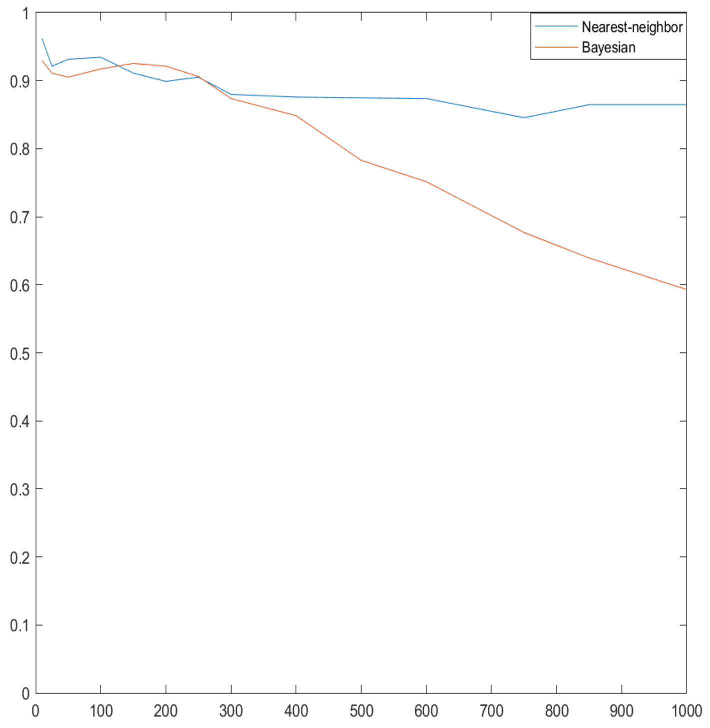


Figure 2 — Classification performance with the Nearest-Neighbor and the Bayesian classifier at different values of K.

By plotting the average accuracy for both types of classification using different values of K, we can observe that in general the Nearest-Neighbor works better for this kind of problem. For values of K ranging between 10 up to around 300, the performance of both classifier is somehow comparable. However, there is a region in which the Bayesian classifier outperforms the Nearest-Neighbor, which is the area ranging between 150 and 250.

Another important remark is that, while the Nearest-Neighbor seems to maintain a stable behaviour even with increasing values of K, after a certain threshold, the Bayesian classifier start decreasing its average accuracy linearly with the number of K.

In general, in terms of computational complexity, the Nearest-Neighbor classifier is much more expensive and it can get very slow for larger datasets. On the other hand, the Bayesian classifier is much more faster and can be applied also to larger datasets. The problem with the Bayesian classifier is that it assumes the features to be independent between each other, and this does not always hold true in real-world applications. This can be seen as a limit of the Bayesian classifier itself, and it mines its performance in most cases.

5 Use your own dataset

For the bonus part of the assignment, I tried the previously implemented image classifier on the CIFAR-10 dataset. CIFAR-10 is a dataset consisting of 32x32 RGB images divided in 10 different classes. Since our task was a binary classification, I selected two classes from this dataset, namely the "dog" class and the "airplane" class. Furthermore, I only considered 300 training images and 50 testing images for each class (the original dataset contains for each class 5000 images for training and 1000 images for testing).

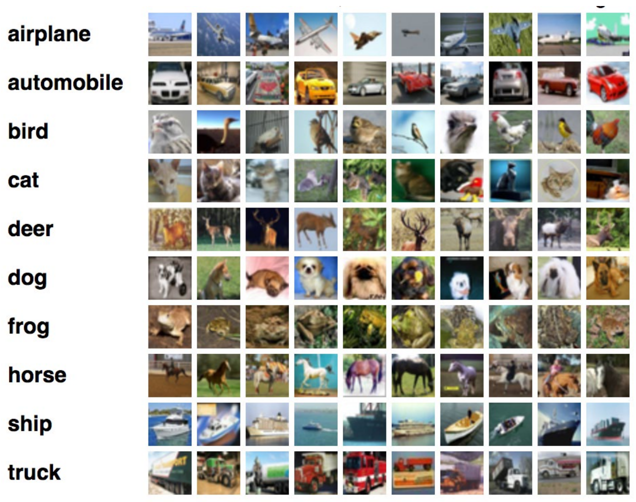


Figure 3 — Schematic representation of the CIFAR-10 dataset.

In this case, since the images have a smaller size compared to the ones previously considered in the assignment, I had to change some parameters for the extraction of local feature descriptors, such as the `cellWidth`, `cellHeight`, `nCellsW`, `nCellsH`, `nBins`, `border` and the length of the vectors `vFeatures` and `vFeatures`. Since these parameters were somehow hardcoded to the function we implemented in the task, I duplicated some of the functions, in which I changed these values, in particular `create_codebook2.m`, `visualize_codebook2.m`, `descriptors_hog2.m` and `create_bow_histograms2.m`.

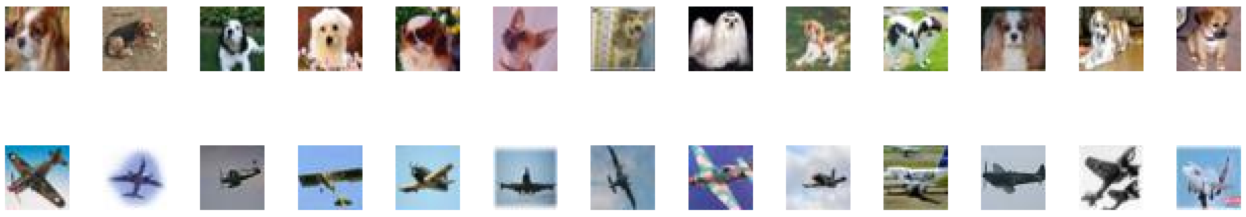


Figure 4 — Examples of images taken from the "dog" and "airplane" classes in the CIFAR-10 dataset.

The results I obtained by setting K = 30 were the following:

K = 30	NN classification accuracy (%)	Bayesian classification accuracy (%)
Iter 1	0.6800	0.7500
Iter 2	0.5700	0.7700
Iter 3	0.6400	0.7200
Iter 4	0.6100	0.6300
Iter 5	0.6800	0.7400
Iter 6	0.5700	0.7500
Iter 7	0.6300	0.7600
Iter 8	0.6400	0.7000
Iter 9	0.6300	0.7500
Iter 10	0.6500	0.7600
Average	**0.6300**	**0.7330**

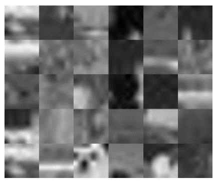


Figure 5 — Codebook generated by setting K=30 in the dog-vs-airplane classification.

The overall performance of the two types of classifiers on this new dataset was worse with respect to the previous dataset. By computing the average accuracy over 10 iterations, I obtained an average accuracy of 63% with the Nearest-Neighbor classifier and around 73% with the Bayesian classifier. Contrary to what was obtained before, in this case the Bayesian classifier seems to work slightly better than the Nearest-Neighbor.

I will now repeat the same experiment by selecting two different classes. I will still consider the "dog" class, but instead of comparing it with the "airplane" class, I will now consider the "cat" class.

This choice was intentionally made since the "dog" and the "cat" classes are more similar between each other (in terms of shape and colors, so it is more likely that they have similar local features), so I expect the performance of the classifier to be much more lower, as it would be more difficult to correctly discern the two cases.

On the other hand, the "dog" and the "airplane" classes had much more different local feature descriptors (in general, dogs and airplanes have way different shapes and textures), therefore the classification could still bring some good result.

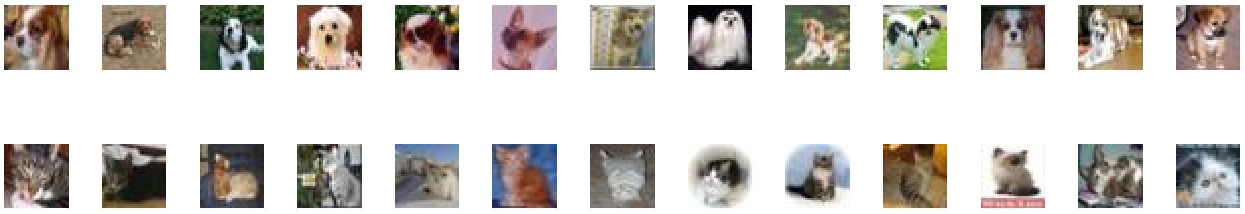


Figure 6 — Examples of images taken from the "dog" and "cat" classes in the CIFAR-10 dataset.

K = 30	NN classification accuracy (%)	Bayesian classification accuracy (%)
Iter 1	0.5200	0.5500
Iter 2	0.5700	0.5800
Iter 3	0.5100	0.5100
Iter 4	0.4500	0.5000
Iter 5	0.4900	0.5700
Iter 6	0.5600	0.5000
Iter 7	0.5400	0.5400
Iter 8	0.5800	0.5400
Iter 9	0.5100	0.5400
Iter 10	0.5000	0.4800
Average	**0.5230**	**0.5310**

In this case, I obtained a very bad classification accuracy. In both cases, the accuracy was around 50%, which basically means that our model is not making any improvement with respect to chance (for a binary classification, having an accuracy of 50% means that we are randomly deciding whether an image belongs to class 1 or 0).