Universitat Politècnica de Catalunya
Universitat de Barcelona
Universitat Rovira i Virgili

Master in Artificial Intelligence

Computational vision

Bag of Words (BOW) for Object Recognition

Authors:
Alejandro Suárez Hernández
Johannes Heidecke

January 2017

Contents

List of Figures

1 Object recognition by BOW

1.1 Classification and feedback

(a) Add a function to visualize the images in a green framework if the classification is correct and in a red framework if it does not

The source code to conduct this task is located at `ex1.m` and `classify_and_show.m`. Basically what we do is to read the `tiny-model.mat` file generated by the script `phow_caltech101.m` which, among other things contain the SVM's weights and offsets to classify the images. We believe that the code is straightforward enough, so we refer the reader to the annex and to the scripts themselves to see more details. We can see some results in figure 1.

(a) *BACKGROUND_google* image incorrectly classified as *Motorbike*

(b) *Faces* image correctly classified

(c) *Motorbike* image correctly classified

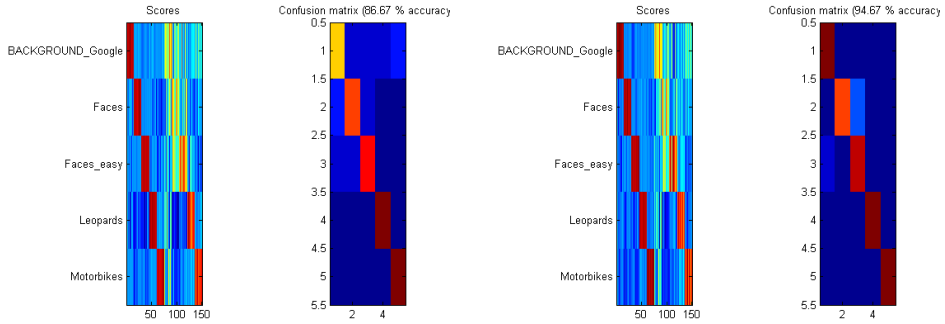
Figure 1: Examples of image classification

1.2 PHOW descriptors

(b) What are the PHOW descriptors? What does the Sizes parameter mean? What does happen if the Sizes parameter is augmented? What does the Step parameter mean? What does happen if the Step parameter is augmented?

PHOW stands for Pyramid Histogram Of visual Words. The PHOW descriptors are those of SIFT, only that obtained with the dense variant of the algorithm. In this variation several points are evaluated in a reduced number of scales. In our particular case (*tiny* problem) we use one scale. The scale(s) is specified with the *Sizes* parameter. The *Sizes* parameter is initially set to 7. Should the *Sizes* parameter be increased, PHOW will extract features that correspond to high scale image structures. We can appreciate the effect of incrementing the scale from 7 to 14 in figure 2. We can observe that the accuracy actually improves. This let us to believe that the most adequate value for the *Size* is the one that

adapts to the structures that best discriminate between images of different categories. Another consequence of changing this parameter is a significant reduction of the (vocabulary) training time, as shown in table 1. Therefore, in this particular case it would seem that setting *Sizes* to 14 is a win-win situation.



(a) Initial results (Size=7, Steps=5) **(b)** $Size = 14$

Figure 2: Effect in terms of accuracy of increasing *Size* from 7 to 14

Elapsed time	**Training vocabulary**	**Testing**
Initial parameters	142.73s	0.57s
Sizes=14	122.21s	0.56s

Table 1: Effect in terms of time of increasing *Size* from 7 to 14

In dense SIFT the key points are chosen from a reduced number of candidate distributed in a grid fashion. The evaluated pixels are separated from each other by a fixed distance, which is the *Step* parameter. In our case, the *Step* is initially set to 5. If we set the *Step* parameter to a higher value, the training (actually the extraction of the vocabulary which is more of a preprocessing step) and the predictions will be significantly faster, but there will fewer candidate pixels for choosing the key points (and therefore, a reduced number of descriptors). We see the effects in terms of accuracy of increasing the *Step* from 5 to 10 and to 20 in figure 3. The accuracy does not get worse until we set the step as high as 20. In addition it is possible to appreciate the notable speedup in the training phase in table 2.

Elapsed time	**Training vocabulary**	**Testing**
Initial parameters	142.73s	0.57s
Step=10	19.07s	0.45s
Step=20	4.00s	0.45s

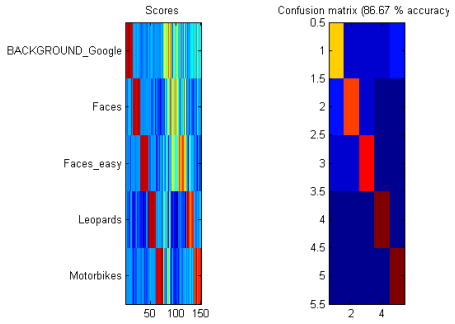
Table 2: Effect in terms of time of increasing *Step*

1.3 Extraction of the vocabulary

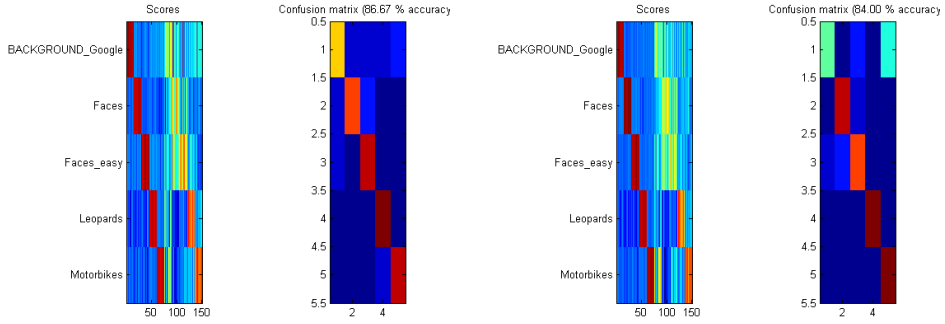
(c) What are the words in the algorithm? How are they extracted? What is their dimension? How their number does affect the accuracy of the results?

The words consist in SIFT descriptors. However, these descriptors are not directly the descriptors extracted from the images. An unsupervised learning procedure receives these descriptors and obtains a set of clusters that summarizes them all. More specifically, the algorithm uses K-means with the Elkan heuristic for reducing the computing effort. The centroids of the obtained clusters are used as the vocabulary words.

Since the words are just SIFT descriptors, their dimension is 128. The effect of the number of words can be understood in terms of the concepts of overfitting and underfitting.



(a) Initial results (Size=7, Steps=5). Acc: 86.67%



(b) *Steps* = 10. Acc: 86.67% **(c)** *Steps* = 20. Acc: 84%

Figure 3: Effect in terms of accuracy of increasing *Step*

Too few and they will not be able to capture all the particular structures that can help in the discrimination between two images (underfitting or high bias). Too many and they will be too specific for the images in the training set (overfitting or high bias).

(d) To construct the vocabulary, the method uses the k-means algorithm. Explore and explain how the k-means is applied, which parameters are used, what is there meaning and what are the advantages and the limitations of the method.

As said before, the k-means algorithm is applied to the descriptors extracted from the images of the train set. The most important parameter (after the descriptors, of course) is the number of desired clusters which is, of course, the number of works we want in our vocabulary. The computational effort of the vanilla K-means is counteracted with the Elkan heuristic that avoids the computation of several distances by application of the triangular inequality. The main drawback of this is that uses storage proportional to the number of clusters[1], so it is not advisable for problems in which we want a high amount of clusters. The use of this heuristic (or any other) is specified as a parameter. Another problem with K-means is that it cannot adapt to clusters with arbitrary forms. An alternative that provides some flexibility in this regard is the well-known EM (Expectation-Maximization) algorithm, which assumes Gaussian clusters with arbitrary covariance matrices. However, this algorithm is more computational intensive and we have already seen that the use of K-means already leads to good results.

[1] `http://www.vlfeat.org/api/kmeans-fundamentals.html`

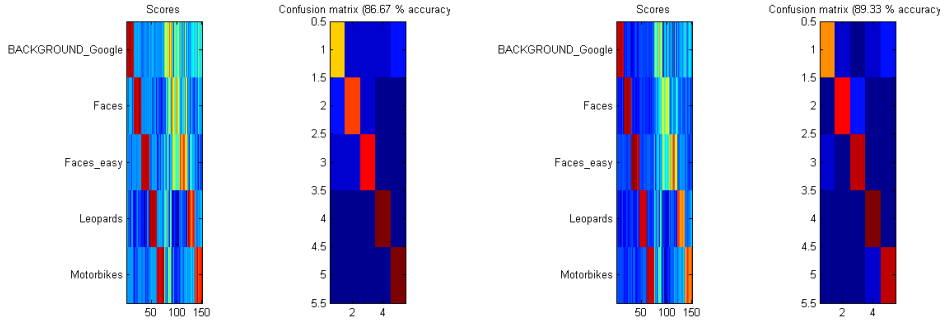
1.4 Spatial histograms: the image descriptor

(e) The function getImageDescriptor gives the spatial histogram of the image. What is the spatial histogram? What dimension does it have? What happens if we augment model.numSpatialX from 2 to 4?

In order to include some knowledge about the location of the visual words in the image, it is divided into smaller quadrants. Initially, the image is divided into 2×2 sectors, as per the *numSpatialX* and the *numSpatialY* parameters. Instead of counting the number of word occurrences in the whole image, we count the number of words in each sector. Therefore, the image descriptor consists of the count of word occurrences for each quadrant. In other words, the total dimension of the image descriptor (which is not the same as the PHOW descriptors) is the number of words times the number of quadrants. This is the spatial histogram and it is what is computed by the *getImageDescriptor* method.

Should we increment *numSpatialX* (or for that matter *numSpatialY*) from 2 to 4, we would multiply by 2 the number of sectors and, consequently, the dimension of the spatial image histogram.

In figure 4 we can see that setting *numSpatialX* to 4 has led to a slight increase in the accuracy.



(a) Initial results (numSpatialX=2) **(b)** $numSpatialX = 4$

Figure 4: Effect in terms of accuracy of increasing numSpatialX from 2 to 4

1.5 SVM classification

(f) The SVM classification with homogeneous kernels is performed by the vl_svmtrain function that needs the homogeneous kernel map obtained by the vl_homkermap function. What are the parameters obtained after the training process? What dimension do they have? How do you apply them to obtain the score of the final classification?

The parameters obtained after the training process are the weight vector (the w) and the offset (the b). The score is obtained performing the inner product between the weights vector and the input transformed by the homogeneous kernel map (the *psix* vector) and adding to that the offset. This is done with a different w and b for each of the classes since we are performing a One vs All classification.

The input data is the spatial histogram, which has a dimension of 1200 (300words $\times$ 4sectors). However the transform maps the data into a higher dimensional space. The *psix* vector has a dimension of 3600. The dimensions of the weight vector w are consistent with this (i.e. also a vector with 3600 rows). Since we are working with 5 classes, we have 5 columns for w and 5 for b.

1.6 Robustness against image transformations

(g) Is the algorithm invariant to rotation of the images? Is the algorithm invariant to rescaling of the images?

If by *rotation of the images* we understand rotation both of the training images and of the test images (or in general any other image that is classified after training the SVM), then sure enough the algorithm will be invariant to rotation (after all this is one of the strengths of the SIFT algorithm that is used to extract the descriptors). On the other hand, if we mean that the test images that are later passed to the classifier are rotated with respect to what the classifier expects, then the classification will fail due the fact that we are using spatial histograms. As long as there are more than one sector (i.e. $numSpatialX > 1 \vee numSpatialY > 1$) the algorithm will be prone to fail in the presence of strong rotations (e.g. a car upside down when the classifier was training with cars normally orientated).

On the other the algorithm is invariant to the rescale of the image (or almost, after all we cannot expect it to work if we scale the image down to one pixel). Again the SIFT descriptors provide us this robustness. In fact, it is worth mentioning that before extracting the spatial histogram, the algorithm scales the image so it has a fixed height of 480.

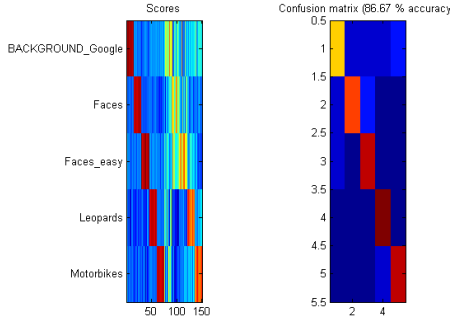
1.7 On the effect of increasing the number of classes

(h) Compute how the accuracy changes when the number of categories augment (e.g. from 5 to 10 to 15 to 100).

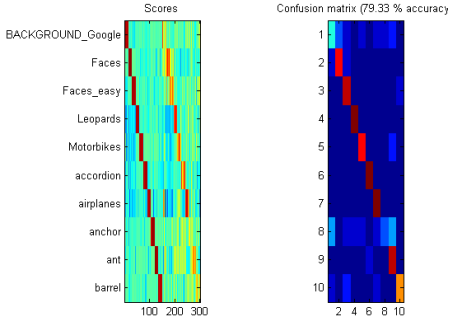
NOTE: in order to perform this check faster, we have set the *Step* parameter to 10. While it does not alter the accuracy when using 5 classes it can affect the results with a higher number of classes. Even so, we believe that the results are still orientative.

As one could anticipate, increasing the number of classes affects negatively the performance of the classifier. The greater the number of classes it has to identify, the more prone it will be to failure. Figure 5 illustrates this issue.

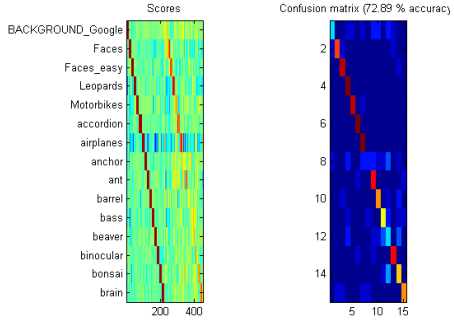
The accuracy in problems for large number of classes has to be optimized by means of tweaking the different parameters that govern the algorithm. For instance, for a larger number of classes one could think that extending the vocabulary, increasing the number of sectors or adding more scales to the PHOW feature extractor is a good idea. This is not covered in the present document.



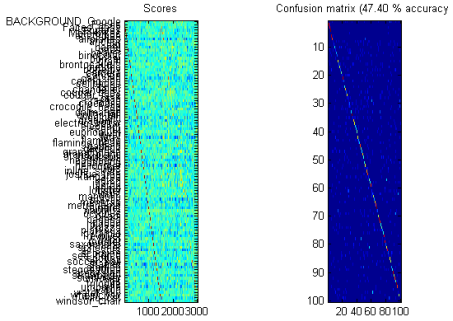
(a) Results with 5 classes (Acc 86.67%)



(b) Results with 10 classes (Acc 79.33%)



(c) Results with 15 classes (Acc 72.89%)



(d) Results with 100 classes (Acc 47.40%)

Figure 5: Effect of increasing the number of classes

Appendices

A Source code

For the present report we have made use of the `phow_caltech101.m` script provided in the `apps` folder of the `vlfeat` toolbox. To extract some of the results presented here, we have modified the script in place.

Apart from that and for the very first part of the report (the image classification and feedback by means of a colored frame) has been done by means of the following functions:

- `classify_and_show.m`: Given a category (as a string), an image file name (e.g. `image_0001.jpg`) and a model classifies the image using the provided model. Then show the image with a green frame if it was correctly classified, and with a red frame if it was not. The title of the image shows what has been ascertained vs the ground truth.
- `ex1.m`: it loads `tiny-model.mat` which contains the parameters of the latest trained "tiny" model (the `phow_caltech101.m` has to be executed first). Then it starts loading and classifying the images from the categories the model has been trained for. It uses the *classify_and_show* method described above.

We refer the reader to the source code for more details.