16-720 Computer Vision **Spring 2016**

Azarakhsh Keipour (akeipour@andrew) **Assignment 1**

Q 1.0 What are the properties of the filter functions?

There are 4 types of filters defined in this project. These filters are trying to catch the "texture" content of the image (scene). The detailed description is as below:

1. *Gaussian filter*: by smoothing (blurring) the image with different intensities, this filter tries to pick up the value of the neighborhood of each pixel in each layer. In other words, the response of this filter on each pixel is the value of the pixel mixed with the values of its neighbor pixels. For example, for a pixel p in L layer (in Lab image) it will catch the light value around pixel p.
2. *LoG (Laplacian of Gaussian) filter*: this is a second order filter and is used to find edges in the image. Therefore, it can pick up the edge information (second-order information) of a pixel.
3. *dx and dy (Derivatives of Gaussian) filters*: these are first-order filters and are used to detect changes in the image along x and y directions, respectively. Thus, these filters pickup changes across a pixel.

Q 1.2 Parameters chosen for K and alpha

- For the number of clusters (K) different values were tested. The results reported in this report are for $K = 100$ which gave best results.
- For the number of random pixels in each image (α) also different values were tested. The results reported in this report are for $\alpha = 150$ which gave best results.

Q 1.3 Computing Visual Words bug

There is a small bug in the given implementation of batchToVisualWords function. The `matlabpool close` command at the last line is not supported by Matlab 2015 anymore, and it should be changed with `delete(gcp('nocreate'))` command.

Intermediate result after step 1

After performing the step 1 (calculation of the word maps), an example image along with its word map is shown in figure 1.

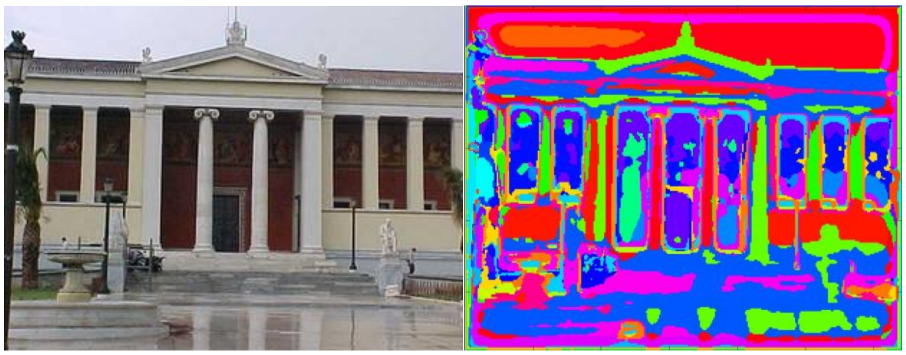

Figure 1. (a) Original picture. (b) Visual Word map of the image shown using *imagesc*.

Q 2.5 Quantitative evaluation

The confusion matrix for a run with K = 100 and α = 150 is as below:

	Airport	Auditorium	Bedroom	Campus	Desert	Football Stadium	Landscape	Rain Forest
Airport	15	1	2	1	0	0	0	1
Auditorium	6	8	2	2	1	1	0	0
Bedroom	7	7	4	1	0	0	0	1
Campus	0	1	0	11	1	1	4	2
Desert	1	0	1	1	11	2	4	0
Football Stadium	0	1	1	3	2	9	1	3
Landscape	4	0	0	3	3	3	6	1
Rain Forest	3	0	1	0	0	0	0	16

We can see that there are 80 images classified correctly and 80 incorrectly which results in 50.00% accuracy.

Q 2.6 About the failed cases

By looking at the confusion matrix above we can easily see that the Rain Forest's accuracy is higher than the others:

Number of Rain Forest images = 20

Number of correctly classified cases (True Positives) = 16 (80%)

Number of incorrectly classified cases (False Negatives) = 4 (20%)

Number of other classes incorrectly classified as Rain Forest (False Positives) = 8

The worst result is for the Bedroom which has only 4 (i.e. 20%) correctly classified (true positive) cases. In 14 cases out of 16 incorrect classifications it is confused with Airport or Auditorium. By looking at these sets of training images one can see the textural similarities in them and this confusion is mostly due to the similarity of the type of Gaussian (and its derivative) filter responses in these classes of images. It seems possible to reduce this confusion by carefully selecting better training images or by running again the dictionary computation.

In the code (`evaluateRecognitionSystem.m`) every failed case is reported along with its correct classification. By examining the errors, you can almost always find the reason of the error: some are misclassified because of their similarity in color to the other classes, some due to the kind of edges, some due to the light conditions, etc. On the other side, those image classes with minimum similarity in color, brightness, type of edges, etc. are never confused (e.g. Auditorium and Rain Forest). It is very hard to correctly classify the images with a very similar filter response to a trained image's pixels.

Q 2.7 Improving performance

Before my custom implementation, I did play with the parameters of the baseline implementation, getting these results:

- By changing K and α the accuracy decreases a little. For example, for K = 200 and α = 100 the accuracy is 41.88%. However, I did not test the values higher than 150 for α (I remained in the range recommended by problem description), but it probably will slightly increase the accuracy while decreasing the performance of building the dictionary.
- I tried running the algorithm without the SPM which resulted in drop of accuracy to 40.00%.
- By increasing the number of layers to 4, the accuracy slightly went down (to 47.50%).

Custom changes made are as below:

- Modified evaluateRecognitionSystem to use multiple cores. With my system (4 cores) the running time reduced from 84 to 57 seconds.
- Using a better similarity measure for the histogram comparison will definitely improve the results. Therefore, I tried many similarity measures for histograms (in

distanceToSet) to find a better one. I used Matlab's pdist2 function. Also, I downloaded the code of chi-squared from the Internet. All the distances are available in the code (distanceToSet.m). Results are:

  - Maximum of element-wise product: Decreased accuracy to about 25%
  - Minimum of sum of squared differences (Euclidean): Decreased accuracy to about 35%
  - Maximum correlation: Decreased accuracy to about 40%
  - Minimum chi-squared distance: The accuracy remained the same 50%
  - Maximum Bhattacharyya coefficient: Accuracy increased slightly to 51.88%
  - Maximum of Spearman's rank correlation: Increased accuracy to 52.5%

- Instead of random α points I used strongest feature points of SURF for a quarter of the points (with 3α/4 random points). In this way I tried to capture important points. The code is included in the getFilterBankAndDictionary.m file. Results of the change are:
  - Using only feature points omits the general scene properties, which can be seen by ~35% accuracy when only using feature points.
  - Using 50% feature points with 50% random points almost gives the same accuracy (~50%) of not using feature points at all.
  - Using α/4 feature points with the remaining 75% points selected at random, gives a slight improvement. With Spearman's rank correlation measure, the result is 53.13%.

In overall the most important improvements are parallelization to improve speed of the evaluation, the change of the similarity measure to a more meaningful Spearman's rank correlation, and the use of some more important points (SURF feature points) with the random ones to improve the accuracy of the system. These customizations to the baseline implementation improved the results from 50.00% to 53.13%.