[EQ2425] Analysis and Search of Visual Data - AY 2019/2020
Project #2 Report

VISUAL SEARCH SYSTEM

Mayank Gulati (gulati@kth.se)
Federico Favia (favia@kth.se)

1. Introduction

The aim of the project is to build a visual search system [1] through the use of a hierarchical k-means tree structure and evaluate it. The features for the images are represented by SIFT keypoint descriptors. In particular, the tree is built based upon a database (server) of 150 images representing 50 different buildings (documents), therefore three images for the same building. For querying a client database of 50 images is used. The score system used for object retrieval in the project is the TF-IDF (term frequency inverse document frequency) scores.

2. Methods

For developing the project, the chosen programming language has been Python 3.7, using the open-source library OpenCV (a modified version with contributions in order to use the patented algorithm of SIFT). The code is composed of a main file (main.py), a file with customized classes (classes.py), a file with the function responsible of building the hierarchical tree (hi_k_means.py) and a open-source library Pytree[5] for tree data structures in Python (treelib.py), modified according to our needs.

3. Image Feature Extraction

The first phase of the project is about extracting features (SIFT keypoints) from the images, both of the database and of query.

(a) It was noticed that in the folder of the database images there were some missing images. Since according to the instructions, they should appear in three similar images for the same building, we have decided to add the missing ones by our own. Therefore, it results in having perfectly 150 database images, three for each building. Few hundred of SIFT features have to be extracted from each image according to the requirements. These features are extracted through the dedicated OpenCV function only keeping the 350 strongest for each image. This number has been decided after a trial and error method since it influences both the computational time and the proper functioning of our tree creation algorithm. Indeed, smaller the number of keypoints to extract, higher the probability to throw a "Zero

Division" error with a deeper tree. For instance, 350 extracted SIFT keypoints from each image are not enough to build a tree with a depth of 7 levels and 5 branches for each level, but our algorithm is able to overcome this issue using exception handling methods (when a empty cluster is found a empty list of TF-IDF scores will be stored).
Then the features of a same building are combined removing the similar descriptors. This removal is performed after performing a SIFT match algorithm based on Nearest Neighbour Ratio Distance between the first image descriptors and the other two corresponding image of the same document. The chosen threshold for the ratio distance is 0.8, similar to the ones reported in literature. Once combined the same features for a building, they are indexed to create the tree later. With these parameters, the average number of features extracted per building (document) is 785.06.

(b) With the same method, 350 SIFT keypoints are detected for each of the 50 images of the query database and they are saved separately. Intuitively, the average number of features extracted per database image should be 350 but in our case is 350.34. This strange value is probably due to some errors of OpenCV SIFT algorithm in detecting a fixed number of keypoints.

4. Vocabulary Tree Construction

In order to build the visual search system, a vocabulary tree [3] based on hierarchical k-means algorithm is chosen as data structure. This structure is controlled by the tree branch number b, and the tree *depth*. In the code, a dedicated functions called *hi_kmeans(_first_node, _des_database list, _b, _depth, _n_documents)* is called to build the tree, whose basic structure is derived from open-source Python library called PyTree. The tree structure is then customized to contain more information to meet the requirements. The parameters passed to the function are respectively:

- The passed first node is created in the main with the descriptors list of the database.
- The descriptor database list is used to do the k-means clustering on it.
- The number of branches in each level is b.
- The number of levels is depth.
- The number of documents, in this project always 50, is passed as well.

The k-means clustering is performed through the already built-in function imported from *sklearn.cluster*.

(a) In order to query by SIFT features, the necessary information to store in each node is the centroid vector of the cluster in that node. This will be used for searching the closest child (computing euclidean distance of each query descriptor from centroids) and follow the optimal path to search the visual word. Furthermore, the entire new cluster with each descriptor vector indexed by the corresponding building (document) is stored in each node, in our case.

(b) In all the nodes the TF-IDF scores table (one score for each of the 50 documents) is stored, but only the one in the leaf nodes will be used for querying. This important parameter is connected with information theory and it refers to term frequency - inverse document frequency (1): it increases the performances of a visual search system instead of only counting.

$$w_{ij} = \frac{f_{ij}}{F} log_2 \frac{K}{K_i} \tag{1}$$

In our scenario, in each node the weight for a certain document w_{ij} is composed by the multiplication of respectively:

- The term frequency ($TF = f_{ij}/F$), different for every building (document), is the number of occurrences f_{ij} of a "building" in that cluster normalized by F the number of descriptors in that cluster.
- The inverse document frequency ($IDF = log_2 K/K_i$) is constant for every document in each node and it is the ratio of the number of documents K to the number of documents that occurs at least one time in the cluster K_i.

(c) Basically, the function used to generate the vocabulary tree [4] is created in such manner. Only if the number of keypoints is greater than the number of branches, it clusters through k-means the SIFT keypoints list and stores these clusters separately. For each of them it computes and saves the tf-idf scores table (as a list), and then it adds these children newly created to the parent node. If there is still depth in the tree, then the function starts recursively calling itself. It is interesting to note that each time the code is run, the tree can slightly change due to the random initialization of k-means clustering.

To refer to the function please open the attached source code (hi_k_means.py).

5. Querying

In the beginning, all the descriptors of each query object (50 images) are sent into the vocabulary tree in order rank the database objects according to their TF-IDF scores. To evaluate the goodness of the system, the average recall rate over the 50 query images is computed, both based on top-1 ranked and top-5.

(a) Three different vocabulary trees are built by changing number of branches (*b*) and depth (*depth*) of the tree and then the querying process is performed. The querying process begins with comparing every descriptor of query image to centroid of the nodes after passing through the root of the tree. The path through the most similar nodes is selected choosing the minimum euclidean distance of the query *q* from the centroids *c* in the same level:

$$Min(\sqrt{\sum (q_i - c_i)^2}) \tag{2}$$

When it reaches a leaf node the corresponding TF-IDF scores table will be accumulated, as well for each other descriptor. At the end, the highest score will correspond to the recognized object (top-1), otherwise the 5 best ranked objects can be found (top-5). For these three trees, the recall rate results are reported below.

Tree parameters	**Average top-1 recall rate**	**Average top-5 recall rate**
b = 4; depth = 3	0.2	0.52
b = 4; depth = 5	0.62	0.8
b = 5; depth = 7	0.8	0.86

First of all, the average top-5 recall rate is of course always higher than the top-1, because the classifier is more relaxed: in fact, it needs only to find the real label in one of the top 5 ranked accumulated TF-IDF scores to recognize correctly the building. The shallower tree is the one which gets the worst results (marginally acceptable only if looking at the top-5 recall rate, 0.52) while the deeper tree achieves an accuracy in the range between 80% and 86%.

(b) In the second part, only the last and deeper vocabulary tree (b = 5, depth = 7) is used. It has been fed with a smaller number of query features (respectively 90%, 70% and 50%). In order to do this, less features are computed every time in each query image and then sent to the tree. The results are shown below.

Percentage of query features	**Average top-1 recall rate**	**Average top-5 recall rate**
90%	0.6	0.82
70%	0.58	0.8
50%	0.48	0.78

The results are reasonable because less features leads to decrease in the accumulation of TF-IDF scores which may affect the correct object recognition.

(c) Hierarchical clustering helps to increase the search speed because it overcomes the issue of calculating as many Euclidean distances as all the descriptors for each database image. In fact after the clustering, where similar descriptors are efficiently grouped together, the Euclidean distance is only computed from the centroids (one in a cluster). In addition, the search algorithm of the tree follows the path only through the closer centroids, avoiding calculations of Euclidean distances from clusters that are not similar.

6. Bonus

One can enhance performance of Bag of visual words by giving importance weights while accumulating TF-IDF scores for different possible paths in the tree. One can also try with alternative methods like RANSAC (Random Sample Consensus)[2] where the geometric transform between query and database image can be estimated by using robust regression techniques.

7. Conclusions

In conclusion, Bag of Visual words provides efficient framework for classification of images using SIFT descriptors as determining features to differentiate images.

References

[1] D. Nister and H. Stewenius Scalable recognition with a vocabulary tree, in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), June 2006

[2] M. Flierl. Analysis and Search of Visual Data [EQ2425], Course Slides, KTH, 2019

[3] Juan Carlos Niebles and Ranjay Krishna Stanford Vision and Learning Lab. Lecture: Visual Bag of Words

[4] University of Florence, Media Integration and Communication Center. Vocabulary Tree

[5] Yoyo Zhou. PyTree - Copyright 2012