EQ2425 Analysis and Search of Visual Data

EQ2425, Project 1

Aleix Espuña Fontcuberta
aleixef@kth.se

Ioannis Athanasiadis
iath@kth.se

September 30, 2021

Summary

In this project, we considered the problem of object retrieval through constructing a visual search system. The search system is performed in two steps. First, we use the SIFT algorithm [2] and the hierarchical k-means algorithm to generate a meaningful low dimensional representation of each image. This process uses the tf-idf score used in document retrieval tasks [3]. The second step is responsible for comparing the representation of a query image, which contains an unknown object, to the server images, whose objects are known. The matching is done according to the closest distance between the different tf-idf representations. We conducted experiments suggesting that the tf-idf approach is indeed capable of accurately representing the images which results in satisfactory retrieval performance. The best results were obtained for the deepest and largest tress combined with the Spearman distance metric.

1 Introduction

In this project we focus on the problem of object recognition via image retrieval. We have a data-set of 50 query images where each image contains an object. The goal is to predict the correct object type. We have 50 different types of objects in total (one unique object per query image). The prediction of the label is done via image matching. That is, we map a query image to a database image whose object type is known. Our database of images contains 150 images. To do the image matching, first we represent each image by a low dimensional vector. The vector is obtained by combining the SIFT algorithm [2] and the hierachical k-means algorithm. Using the vector representation, we can compute the distance of a query image with respect to each of the database images. We match the query image to the closest database image.

2 Problem Description

The most challenging part of the project is to generate an informative low-dimensional representation for each image. This proceeds in the following manner. At first, the SIFT algorithm is employed to each database image, which produces a large set of image descriptors. Then the image descriptors are used to generate a tree with the hierarchical k-means algorithm. Building the tree requires specifying its depth as well as its branch factor. The database descriptors are hierarchically clustered into groups using the hierarchical k-means algorithm. More specifically, at each node, the k-means is applied on the descriptors while the latter are passed to the children corresponding to their assigned cluster. The

final result is a set of nodes, where each node i contains a cluster centroid $\boldsymbol{\mu}_i$. The ultimate destination of each clustered descriptor is a tree leaf (a node without children). From this tree, a low dimensional vector can be built to represent each database image or query image. The vector will have as many components as tree nodes. We followed the procedure described in [1]. Using their notation, a database image j is described by a vector $\mathbf{d}$, where a component $d_i = m_i w_i$. The variable m_i is named term frequency and it is the number of descriptors of the database image j that went through node i. The variable w_i is known as inverse document frequency, and it has the expression $w_i = \log_2 \frac{N}{N_i}$. Where in our case N is the number of database objects (50) and N_i is the number of objects that reached node i. We note that this representation is known as tf-idf weighting [3], and it is used for document retrieval tasks in the field of natural language processing. Finally, the query vector $\mathbf{q}$ used to represent a query image is designed similarly. Its component is $q_i = n_i w_i$. Where n_i is the number of descriptors of the query image that went through node i. Once the vectors $\mathbf{q}$ and $\mathbf{d}$ are created, the similarity $S(\mathbf{q}, \mathbf{d})$ between a query and a database image is calculated according to (1). Where the L1 norm is used for normalizing each vector and we will experiment with different norms for the distances between $\mathbf{q}$ and $\mathbf{d}$.

$$S(\mathbf{q}, \mathbf{d}) = || \frac{\mathbf{q}}{||\mathbf{q}||_1} - \frac{\mathbf{d}}{||\mathbf{d}||_1} || \tag{1}$$

3 Results

3.1 Image Feature Extraction

At the feature extraction stage, we are using the SIFT algorithm to generate a few thousand features for each image. Given that we had multiple instances of the same object in the database, we merge the features according to the object they were derived from. More specifically, we used the configuration of 0.005 and 3 for the PeakThresh and EdgeThresh parameters. The mentioned configuration resulted in an average of 13968 and $4495,2$ SIFT features per object for the server and the client respectively.

3.2 Vocabulary Tree Construction

When it comes to the vocabulary tree construction, storing the clusters' centroids at each node is necessary for querying the images. Additionally, in order to compute the TF-IDF score, we need to store an inverted file in each leaf. An inverted file of a leaf contains the list of objects that ended in the leaf. For each object, it also contains the number of descriptors that reached the leaf. We notice this is the variable m_i defined previously.

3.3 Querying

3.3.1 Experiments for different tree depths

After having implemented the vocabulary construction routine, we experimented we different tree setups. In the table below we report the top-1 and top-5 recall rates for the configuration specified in the assignment's description.

branch factor	depth	top-1 recall	top-5 recall
4	3	2%	10%
4	5	74%	86%
5	7	82%	88%

Table 1: Table of top-1 and top-5 recall rates using 100% of the client's features

When employing a tree of branch 4 and depth 3, we observe a recall rate corresponding to random retrieval. This behavior is justified by the idf scores. More specifically, in our case we are using many features and thus a small-depth tree gives rise to 0 idf scores. In other words every cluster contains features derived from all objects. Therefore, the generated representations are not informative at all, which results in random retrieval. When deeper and wider trees were employed, the issue described above was mitigated. For instance for a branch of 4 and depth of 5 we achieved a top-1 recall of 74% and top-5 recall of 88%. When tree of branch 5 and depth 7 was employed, we achieved the highest performance of 82% and 88% top-1 and top-5 recall respectively.

3.3.2 Experiments for a fixed depth

Additionally, we have also experimented with using fewer number of features when quering the client images. Intuitively, this result in having less informative representation of the client images and this is mainly the reason we observe an increase in the recall rates in Table 2. More specifically, the top-1 recall achieved was 82%, 82%, 80% and 72% when using 100%, 90%, 70% and 50% of the client features respectively. Considering that we extracted a relatively large number of features, the observed decrease in the performance is reasonable. When it comes to the top-5 performance, we achieved recall rates of 86%, 88%, 86% and 84% when using 100%, 90%, 70% and 50% of the client features respectively. In the case of top-5 recall, the decrease is less evident, which make sense, since top-5 recall rate is more forgiving compared to the top-1. However, we observed a somewhat strange behavior with having a top-5 recall rate increased by 2% when 90% of the client feature were used compared to when 100% of them were used. One reasonable explanation would be, that from a point onward, using more features does not provide any added value in the representation and might results in some slight confusions during the retrieval process. On the other hand 2% is quite small difference and thus we could also be accounted to randomness as well.

client-percentage	top1-1 recall	top-5 recall
100%	82%	88%
90%	82%	88%
70%	80%	86%
50%	72%	84%

Table 2: Table of top-1 and top-5 recall rates for a tree with depth 7 and branch factor equal to 5 for different number of client features. The similarity metric used is the L1 norm.

3.3.3 About the search speed

Let's consider a single query descriptor. A tree with depth d and branch factor b contains b^d leaves. However, thanks to the tree structure, assigning a leaf to the query descriptor just takes $b \times d$ euclidean distance calculations. This is because at each level of the tree we just have b candidates, the number of children of a

node, and therefore we do not look at all the other nodes, which saves lots of computations. This is different than having a big visual vocabulary and having to compute the distance of the descriptor to each of the visual words.

3.4 Bonus

At this stage we employed additional distance metrics to measure the disimilarity between the server and the client images. More specifically, we experimented with the cosine similarity and the Spearman metric. We report the recall performance for the 5 branch factor and 7 depth configuration in the tables below.

client-percentage	branch factor	depth	L1-norm	Cosine	Spearman
100%	5	7	82%	82%	94%
90%	5	7	82%	82%	94%
70%	5	7	80%	82%	90%
50%	5	7	72%	82%	90%

Table 3: Table of top-1 recall rates for different number of client features and metrics.

client-percentage	branch factor	depth	L1-norm	Cosine	Spearman
100%	5	7	86%	94%	98%
90%	5	7	88%	94%	98%
70%	5	7	86%	94%	98%
50%	5	7	84%	94%	96%

Table 4: Table of top-5 recall rates for different number of client features and metrics.

From Tables 3 and 4, we conclude that the Spearman distance performs the best out of all three metrics in terms of recall rate for both top 1% and top-5% under variant percentage of client features used. When it comes to the cosine similarity metric, we identified it to be quite robust when changing the number of features used in the query documents. We note we achieved the same recall rates of 86% for all 4 different percentages of client features used. Finally, the naive L1-norm performs the worst.

4 Conclusions

Concluding, we observed that the tree of higher branch (5) and depth (7) gave the highest recall rates. Interestingly, we also observed that constructing a tree using small depth and large number of features could potentially result in random retrievals due to having the idf coefficients being 0 and thus generating uninformative queries. Regarding the performance for different metrics, Table 4 suggests that the Spearman metric performs the best. However, we noticed that the cosine similarity metric is very robust under varying number of client features. Finally, we can safely conclude, that the querying by the tf-idf approach is capable of generating meaningful vector representation for the various images and consequently results in competent object retrieval.

Appendix

Who Did What

We both iterated through the papers multiple times and discussed the concept of object retrieval via visual search. When it comes to the coding part, Ioannis implemented the feature extraction module. Regarding the implementation of the Vocabulary Tree Construction and querying, we both worked on that separately and compared our results. In canvas we uploaded the implementation that was more efficient. Finally, we have jointly written the report with Aleix focusing on describing and formulating the problem and Ioannis presenting and describing the results.

References

[1] Nister, David, and Henrik Stewenius. *Scalable recognition with a vocabulary tree. 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*

[2] Lowe, David G, *Object recognition from local scale-invariant features. Proceedings of the seventh IEEE international conference on computer vision. Vol. 2. Ieee, 1999*

[3] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval, Cambridge University Press. 2008.*