# EQ2425 Analysis and Search of Visual Data

EQ2425, Project 2


| Wenrui Zhao | Jingxi Huang | Letao Feng |
|---|---|---|
| wenruiz@kth.se | Jingxi@kth.se | letao@kth.se |


September 29, 2024


## Summary

The goal of the project is to develop a visual search system for retrieving buildings based on query images using SIFT descriptors. A vocabulary tree, constructed with hierarchical k-means, organizes the database, and TF-IDF scoring ranks the similarity between query and database objects. The process involves extracting SIFT features, building the tree with various configurations, querying the tree, and calculating recall rates to evaluate the system's accuracy with different tree structures and partial features.


## 1 Introduction

In this project, we aim to develop a robust visual search system capable of recognizing buildings through the use of image features and advanced data structures. Leveraging Scale-Invariant Feature Transform (SIFT) descriptors, we extract distinctive features from images that effectively capture the essence of each building. These features are then organized within a hierarchical structure known as a vocabulary tree, which enables efficient storage and retrieval.

## 2 Problem Description

### 2.1 Image Feature Extraction

In this section, our task is to extract SIFT features from both database and query images. For the database images which are stored in the server folder, there are 50 building objects in total and each object includes three images. The query images in the client folder contain 50 building objects which are exactly the same as in database. Each object includes only one image.

In order to reduce the calculation time, we extract 1000 SIFT features from each image. We then merge the features of the same object and store them for the construction of vocabulary tree.

### 2.2 Vocabulary Tree Construction

In this section, we need to build the Vocabulary Tree to representing and managing the image feature data effectively. To create this tree, we use the hierarchical k-means algorithm. This algorithm helps in clustering the image features, thus we can form a tree structure. In the tree, each node represents a set of similar features, and each child node further divides these features until the end of the tree (leaf node).

The structure of the tree is controlled by two parameters: the number of branches of the tree (b) and the number of layers in the tree (tree depth). The number of branches of the tree (b) means how many sub-nodes each node can be divided into. And the number of layers in the tree means the maximum path length from the root node to the leaf nodes.

We need to store necessary information in each node. This information will be used later for querying and image matching.

### 2.3 Querying

In this section, we input all the feature descriptors of each query object (client) into the vocabulary tree, and then rank the objects in the database (server) according to their TF-IDF scores. We need to test all the 50 query objects and calculate the average recall rate. Here the recall rate is based on the object. If the object is found, the recall rate is 1. If it is not found, the recall rate is 0.

First, we extract SIFT features from the query image. These features are mapped to the corresponding leaf nodes through the vocabulary tree.

Next, we calculate the TF-IDF score. The TF (Term Frequency) means the frequency of occurrence of a certain visual vocabulary (leaf node) in a single image. And the IDF (Inverse Document Frequency) means how often a visual vocabulary occurs in the whole database. The higher the IDF value, the rarer and more distinguishable the visual vocabulary is. Then, the TF of each visual vocabulary is multiplied by the IDF to get its TF-IDF score. Higher TF-IDF scores indicate that the visual vocabulary is more discriminative of the current image.

The formula for TF-IDF is:

$$TF - IDF = TF * IDF = TF * \log(\frac{N}{DF}) \tag{1}$$

where N is the total number of images in the database, and DF is the number of images that contain the visual vocabulary.

Finally, we rank the database images based on the TF-IDF score. A higher similarity score indicates more similarity to the query image.

# 3 Results

## 3.1 Image Feature Extraction

We extract 1000 SIFT features from each database image and the average number of SIFT features per server object is 2980.72. That is because object 26 and object 38 only contain two images and object 37 contains four images.

Similarly, we extract 1000 SIFT features from each query image and the average number of SIFT features per client object is 1000.24.

## 3.2 Vocabulary Tree Construction

To answer Problem (a), we think each node in the vocabulary tree should store Cluster Centroid, Child Nodes, Number of Features, and Visual Vocabulary ID. The Cluster Centroid means the centroid of the SIFT features that are assigned to this node. It represents the "average" feature of the cluster. Child Nodes allow downward traversal of the vocabulary tree. The number of child nodes is equal to the number of branches b. The Number of Features means the count of the SIFT features that were clustered into this node. The Visual Vocabulary ID is the index that helps link this node to a specific visual vocabulary.

For Problem (b), when considering TF-IDF scores, additional information to store in the leaf nodes includes Term Frequency (TF), which is a dictionary mapping each unique feature (or visual word) to its frequency within that leaf node, allowing for the calculation of TF for each feature associated with the leaf. Additionally, Document Frequency (DF) counts how many different database objects (images) have contributed to the features in this leaf node, assisting in the later calculation of IDF. Inverse Document Frequency (IDF) should also be stored; while IDF can be calculated dynamically, storing pre-computed IDF values in the leaf nodes can speed up retrieval, providing a mapping of each feature to its IDF score across the entire dataset.

For Problem (c), the hi_kmeans function constructs a vocabulary tree using the hierarchical k-means algorithm. It recursively splits the input data into clusters, forming a tree structure where each node represents a cluster center. The function stops splitting when it reaches the set depth or when the number of data points is too small to form further clusters.

## 3.3 Querying

In this section, we need more functions to test all 50 query objects and calculate the average recall rate.

The traverse_tree function traverses the vocabulary tree with a given descriptor to find the appropriate leaf node. This function selects the closest child node by calculating the distance between the descriptor and the cluster center of each child node, and then continues traversing until it reaches a leaf node.

The tf_idf_ranking function computes the TF-IDF scores for each database (server) object and ranks them.

The calculate_recall function calculates the recall rate for a single query object. It uses the results from the tf_idf_ranking function to rank the database objects and checks if the correct object index is among the top top_k results.

The test_vocabulary_tree function tests the recall rate for a vocabulary tree over multiple queries. It iterates through all query objects and computes the Top-1 and Top-5 recall rates, returning the average recall rates for all queries.

In summary, we build vocabulary trees using the hi_kmeans function and tests the recall rates on the object_features_client_list and object_features_server_list.

The results of Problem (a) are shown in the table 1:

| | branches | depth | Average top-1 recall rate | Average top-5 recall rate |
|---|---|---|---|---|
| Tree 1 | 4 | 3 | 0.04 | 0.16 |
| Tree 2 | 4 | 5 | 0.52 | 0.74 |
| Tree 3 | 5 | 7 | 0.80 | 0.92 |

Table 1: Performance of different tree configurations

Similarly, The results of Problem (b) are shown in the table 2:

| Percent of the number of query features | Average top-1 recall rate | Average top-5 recall rate |
|---|---|---|
| 90 | 0.78 | 0.92 |
| 70 | 0.76 | 0.90 |
| 50 | 0.76 | 0.90 |

Table 2: Performance of different percents of the number of query features

For Problem (c), Hierarchical Clustering can significantly reduce the search space and computation. In normal k-means clustering, for each query feature, we need to compute its Euclidean distance from all the clustering centers, and then select the nearest clustering center. If there are N clustering centers, the total number of distance calculations is N. In hierarchical clustering, each level of the tree assigns features to smaller subsets. Assuming that the depth of the tree is d and each node has b branches, then only b distances need to be computed at each level, and then proceed to the next level based on the nearest node. The total number of distance calculations is $b \times d$.

### 3.4 Bonus

We use a smoothed TF-IDF formula

$$TF - IDF = (1 + \log(TF)) * IDF \tag{2}$$

to increase recall rates. In the original TF-IDF computation, frequently occurring features can disproportionately influence the results, potentially overshadowing important features that may occur less frequently. By introducing smoothing in the TF calculation, the influence of high-frequency features on the final weight is reduced while enhancing the relative weight of low-frequency features. We set the parameters of the tree as b = 5, depth = 7 and apply the smoothed TF-IDF formula. We then obtain an accuracy of 0.82 on top-1 recall, and of 0.92 on top-5 recall, which is slightly better than the original result.

## 4 Conclusions

By implementing the visual search system, we built three vocabulary trees with different branch numbers and depths. We tested the system's performance by calculating the average top-1 and top-5 recall rates over 50 objects for each configuration. The results demonstrated that increasing the depth of the tree and the branch factor improves the retrieval performance. Additionally, querying with partial features showed a slight decrease in recall rates, but the system remains relatively accurate even with reduced input data.

# Appendix

## Who Did What

| | contribution |
|---|---|
| Letao Feng | 33.3...% |
| Jingxi Huang | 33.3...% |
| Wenrui Zhao | 33.3...% |

# References

[1] Markus Flierl, EQ2425 Analysis and Search of Visual Data, *Lecture Slides*, 2024