EQ2425 Analysis and Search of Visual Data
Project 2 Report: Visual Search System

Jingxuan Mao jmao@kth.se
Yuqi Zheng yuqizh@kth.se
Aarati Medehal medehal@kth.se

October 2, 2022

Summary

The following Project focuses mainly on building and evaluating a Visual Search System, using SIFT descriptors and vocabulary trees.

The goal of the project is for the visual search system, to be able to recognize the buildings that are present in the database with the most similar building from the Query images. In order to obtain this 'object retrieval', the TF-IDF score is used.

Term Frequency-Inverse Document Frequency is a popular measure used in text retrieval and analysis to help evaluate how relevant a certain 'word' is to document in a set of documents. It is a product of Term Frequency(TF) and Inverse Document Frequency(IDF).

Thus, essentially this TF-IDF score helps to determine how important a word is. Higher the score, the more relevant. Thus, in this situation our 'words' are image features and the 'documents' are images.

Note that the following project has been implemented on Python 3.7 using the OpenCV library to perform feature extraction and related operations on the provided images. Additionally, notable modules that have been incorporated for efficient implementation are as follows:

1. *pickle*, compress the data into a character stream to store all the objects data to a file, so that this information can easily be used further in another function with all necessary information.
2. *KMeans*, iteratively divides the available data into K clusters, with the centre being the average value of all points in that cluster.

1 Image Feature Extraction

The data provided are in the form of 2 folders, Server and Client.

The Client folder consists of our query images which we use to 'query' against the Server folder which is composed of the database images to perform image retrieval.

1.1 Results and Discussions

1.1.1 The first question required us to extract a few thousand SIFT features from each database image, combine the features of the same object and save. Finally, we are required to report the average number of features we have extracted per object

The exact number of database images is 149, comprised of 50 different buildings in either 2,3 or 4 of them. In our case, we have extracted 2000 SIFT features from each database image. On completion, all the descriptors obtained are stored in the form of a matrix, where each row represents a feature descriptor, with the label of each object attached to the last element in the descriptor.

1.1.2 Next, we are required to perform the same feature extraction, but this time on the Query Images(Client Folder).

Similarly, the number of SIFT features extracted has chosen to be 2000 from each different building image.

2 Vocabulary Tree Construction

2.1 Realization of hierarchical k-means algorithm

Hierarchical k-means algorithm is used to build the vocabulary tree, with its realization *hi_kmeans* designed in a recursive manner to grow the tree from its root node all the way down to final leaf nodes. We implement the in-build function *Kmeans* from sklearn to help with the clustering.

The vocabulary tree for the project is fed with a total of 2000*149 feature vectors to be separated and organised into a tree. This organisation is carried through b(branch number) and d(depth). The total number of leaf nodes as well as final visual vocabularies would thus be b^d.

The code of function *hi_kmeans* is attached below:

```
def hi_kmeans(data, b, depth, icenter, ileaf, center):
    '''
    :param data: The SIFT features from the database objects
    :param b: The branch number of vocabulary tree
    :param depth: The number of levels of the vocabulary tree
    :param icenter: The center of all clusters that is divided in
                                    every level
    :param ileaf: The leaf nodes a.k.a. visual vocabularies of the
                                    tree
    :param center: A temporary variable to store the centers of the
                                     clusters grown by a father
                                    cluster
    :return: icenter, ileaf
    '''
    if depth > 0:
        if data.shape[0] < b:
            icenter.append(center)
            for cnt in range(b):
                hi_kmeans(data, b, depth - 1, icenter, ileaf,
                                                center)
        else:
            kmeans = KMeans(n_clusters=b, random_state=0).fit(data[
                                                :, :128])
            label = kmeans.labels_
            center = list(kmeans.cluster_centers_)
            icenter.append(center)
```

```
            for clus_idx in range(b):
                data_in_clus = [m for m, n in enumerate(label) if n
                                                    == clus_idx]
                hi_kmeans(data[data_in_clus], b, depth - 1, icenter
                                                    , ileaf, center)
    else:
        ileaf.append(data)
```

2.2 Information stored in nodes

For the need of querying, we only need to store the information of the cluster center of each node to compare the Euclidean distances between the descriptors of the query image and the database images. But considering the calculation of TF-IDF weights, for leaf nodes both centers and all inside SIFT descriptors with label should be registered to see how different visual vocabularies appear in all 50 objects.

2.3 Vocabulary Tree Construction

In this section, we call function *hi_kmeans* and adjust parameters b and *depth* to build vocabulary tree of different sizes. As a result, three kind of trees are constructed: $b = 4, depth = 3$, $b = 4, depth = 5$ and $b = 5, depth = 7$.

3 Querying

In this section, we utilize the hierarchical vocabulary tree to match the query image with the database object according to the distance of their descriptors.

3.1 Realization of query algorithm

We classify each descriptor of the query image level by level, recursively. In each level of the tree, we select the node with the minimum Euclidean distance from the query descriptor as the classifying centroid of the next level. To implement the recursive query, we need to calculate the absolute location of each centroid in the whole tree. In addition, we need to store the relative positions of these centroids and obtain the final class of visual vocabularies according to them.

3.2 Comparison between result of three different trees

For these three trees mentioned above, we query them with all client images and calculate the average top-1 and top-5 recall rates over 50 objects. The results are shown below:

Table 1: Recall rate of trees with 3 different structures

Recall rate	b=4,depth=3	b=4,depth=5	b=5,depth=7
top-1	0.10	0.74	0.84
top-5	0.36	0.80	0.90

With the tree becoming larger and deeper, both the top-1 and top-5 recall rate are getting higher. This is because the larger and deeper the tree is, the more distinctive each leaf will be, and the finer the classification of the descriptors could achieve. In other words, the entropy will decrease as the tree grows. However, the tree cannot be arbitrarily complex to avoid overfitting.

3.3 Comparison between result of different amount of query data

Now we use vocabulary tree with $b = 5, depth = 7$ and select only 90, 70 or 50 percent of the query features for query. The results are shown below:

Table 2: Recall rate of b5d7 tree with different amount of query data

Recall rate	50% of query data	70% of query data	90% of query data
top-1	0.80	0.82	0.88
top-5	0.90	0.90	0.92

We only query once and see that the recall rate of top-1 is increasing with more query data coming in, which makes sense since the system will work better with more information of the image captured. For top-5 recall rate there hasn't been much increasing because it has already taken more possible candidates into consideration and thus being more robust. Also, The recall rates in 90% case are higher than those of the query with the full set, which happens because the wrong classified features are not selected within, since we only run the program once and randomly choose data and most of them are correctly classified.

3.4 Explanation on the search speed improvement of hierarchical clustering

Take the $b = 5, depth = 7$ tree as an example. With a hierarchical structure, we only need to calculate the Euclidean distance between the query descriptor and each five centroid node descriptor in each level, and select node corresponding to the minimum distance as the centroid of next level, and then, we calculate the Euclidean distance between the query descriptor and five nodes under that centroid. We do not need to consider the nodes under other four centroids anymore. The process above will be repeated until the last level of the tree, i.e. the 7th level, is reached and we will obtain the final classification result. In this process, in each level, we only need to execute 5 Euclidean distance calculations, with a total of $5 \times 7 = 35$ Euclidean distance calculations needed. While without a hierarchical structure, in the worst case, we need to search through all clusters to find the correct one, which means that the Euclidean distance needs to be calculated $5^7 = 78125$ times. The complexity is exponentially increased and thus the search speed will be significantly reduced. Therefore, the hierarchical structure plays an import role in classification.

3.5 Bonus

Since images can be regarded as poses if the object, we can utilize Random Sample Consensus (RANSAC) to implement the geometric transform between the query image and the database image, which enables data smoothing and thus leads to a more accurate classification.

In addition, we can optimize the scoring mechanism. Since the k-means algorithm selects centroids randomly, the constructed tree would differ. We can take the average of the TF-IDF weights of different trees as the final score.

4 Conclusions

This project thus helps to provide a clear picture on how to implement the visual search. Firstly, we extracted the features of the database and query images. Then, we constructed a hierarchical k-means tree with different visual

vocabularies containing different database features. Next, we categorized the features of each query image into those visual vocabularies according to the Euclidean distance between the descriptors. We calculated the TF-IDF weights of the database and query image, respectively, and select the object with the minimum L1-norm distance between the query and database weights as the classification result.

The hierarchical k-means algorithm could achieve accurate classification with TF-IDF weights. Larger and deeper trees had a better performance with appropriately more query data fed in. Overall, this algorithm has a significant advantage in terms of classification accuracy, speed, and robustness, with a great potential for practical applications.

Appendix

Who Did What

Jingxuan Mao and Yuqi Zheng were involved in the major part of the coding aspect and all 3 members contributed in equal amounts along with helping to write the report.