Research Proposal: Optimize the localisation in efficiency and storage

In our previous work, we adopt a brute force search approach in the localisation. The storage and searching time grow faster with longer route length. Therefore, it’s necessary to discard some routes at each time step. We already did a preliminary test of this idea in our ECCV paper by culling 50% routes that have low matching scores at each time step. It shows potential of optimising the localisation with shorter candidates list. However, this kind of method is naive with manually setting parameters *N*. I intend to investigate this problem further with effective learning approaches. This problem is very similar to the search performance optimization of retrieval tasks, in which we could explore some solutions.

The optimization methods are usually divided into two directions: one is search optimization, which is achieved by optimizing the retrieval structure without changing the vector itself. The other is vector optimization, which is achieved by mapping high dimensional floating-point vector to the Hamming space to reduce the complexity of distance calculation. In practical, these two methods are often combined.

1. Search Optimization

The ultimate goal of the retrieval task is to return the result that is most similar to the query, which is usually divided into nearest neighbor search (NN) and approximate nearest neighbor (ANN) search. Compared with NN, ANN can greatly improve retrieval efficiency and find matching targets that approximate the closest distance. There’re already many different kinds of ANN approaches, including Local Sensitive Hash (LSH) [1], Inverted Multi-Index (IMI) [2], Non-Orthogonal Inverted Multi-Index (NO-IMI) [3]. By performing the hash mapping on all data in the original data set, LSH [1] get a hash table, in which similar data are scattered into the same bucket. The hashing strategy is also used in [4] to realize the combination of ultra-compact place representations, near sub-linear storage scaling and extremely lightweight compute requirements.

IMI and NO-IMI are both developed from inverted index [5]. The inverted index structure divides the search process into two parts, index search and distance reranking. Index search generally uses an exhaustive method, traversing to obtain the index corresponding to the query vector, and then obtaining the candidate list for reranking. After calculating and reordering the distances between the query feature vector and the candidate list vectors, the nearest neighbor will be returned. The IMI [2] method needs to ensure that the multiple data sets divided by the feature vector are irrelevant. For traditional features such as SIFT, this condition is met. However, the deep feature does not have the above-mentioned separable conditions, and the divided data space has a strong correlation, so the application of IMI to the deep feature has limitations. An efficient indexing method for massive deep feature data is given by NO-IMI [3].

In most cases, these methods consist of four main steps: feature extraction, dimensionality reduction, feature clustering and feature quantization. They are always achieved separately. Inspired by deep clustering technology [6], it’s possible and attractive that an end-to-end network can be designed to simultaneously get deep features and clusters.

1. Vector Optimization

The search optimization method introduced above improves the search efficiency by reducing the search space. Another way to optimize retrieval performance is to map high-dimensional floating-point vectors to other spaces, mostly Hamming space. In this case, the mapped vector can be calculated in a more efficient manner. There’re already some works realize the vector optimization with deep hashing networks [7-10]. Furthermore, some deep cross-modal hashing networks are designed to correlate images with texts [11-14], similar to our tasks (correlate images with maps). Especially in [7], a hierarchical search is designed to retrieve similar images using a coarse-to-fine strategy that utilizes the learn hashes-like binary codes and F7 features.

In summary, inspired by above methods, we can either use deep hashing or deep clustering to cluster and quantize the float-point features to get shorter candidate lists at each time step. Based on this strategy, the localisation can be optimized in efficiency and storage.

References

[1] p-stable LSH: locality-sensitive hashing scheme based on p-stable distributions, SCG 2004.

[2]. The inverted multi-index, TPAMI 2015.

[3]. Efficient indexing of billion-scale datasets of deep descriptors, CVPR 2016.

[4]. Fast, Compact and Highly Scalable Visual Place Recognition through Sequence-based Matching of Overloaded Representations, ICRA 2020

[5]. Video google: a text retrieval approach to object matching in videos, ICCV 2003.

[6]. <https://deepnotes.io/deep-clustering>

[7] Deep Learning of Binary Hash Codes for Fast Image Retrieval, CVPR workshop 2015

[8] Deep Supervised Hashing for Fast Image Retrieval, CVPR 2016.

[9] Deep Hashing Network for Efficient Similarity Retrieval, AAAI 2016.

[10] Feature Learning based Deep Supervised Hashing with Pairwise Labels, IJCAI 2016.

[11] Deep Cross-Modal Hashing, CVPR 2017

[12] Pairwise Relationship Guided Deep Hashing for Cross-Modal Retrieval, AAAI 2017

[13] Self-Supervised Adversarial Hashing Networks for Cross-Modal Retrieval, CVPR 2018

[14] Self-constraining and attention-based hashing network for bit-scalable cross-modal retrieval, Neurocomputing 2020